İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

FİZİKSEL KLONLANAMAZ FONKSİYONLAR YARDIMIYLA ANAHTAR ÜRETİMİ VE LİSANS DOĞRULAMA

YÜKSEK LİSANS TEZİ

Şahin BAŞ

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

OCAK 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

FİZİKSEL KLONLANAMAZ FONKSİYONLAR YARDIMIYLA ANAHTAR ÜRETİMİ VE LİSANS DOĞRULAMA

YÜKSEK LİSANS TEZİ

Şahin BAŞ
(504111219)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Müştak Erhan YALÇIN

OCAK 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504111219 numaralı Yüksek Lisans Öğrencisi **Şahin BAŞ**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"Fiziksel Klonlanamaz Fonksiyonlar Yardımıyla Anahtar Üretimi ve Lisans Doğrulama"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Müştak Erhan YALÇIN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. İlker BAYRAM** İstanbul Teknik Üniversitesi	.....
	Yrd. Doç. Dr. Selçuk BAKTIR Bahçeşehir Üniversitesi	.....

Teslim Tarihi : **15 Aralık 2014**
Savunma Tarihi : **20 Ocak 2015**

Aileme,

ÖNSÖZ

Bu çalışmayı gerçekleştirmemi sağlayan danışman hocam Prof. Dr. Müştak Erhan Yalçın'a teşekkür ederim.

Ayrıca katkılarından dolayı Doç. Dr. Berna Örs Yalçın'a, Yük. Müh. Ramazan Yeniçeri'ye, Yük. Müh. Emre Göncü'ye ve arkadaşım Seyhan Çalışkan'a teşekkür ederim.

Son olarak yüksek lisans çalışmalarım boyunca beni destekleyen MKR-IC ve TÜBİTAK BİDEB'e teşekkürü bir borç bilirim.

Ocak 2015

Şahin BAŞ
Elektronik Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

AES **:** Advanced Encryption Standard
ASIC **:** Application Specific Integrated Circuit
CLB **:** Combinational Logic Block
ECC **:** Error Correction Codes
FPGA **:** Field Programmable Gate Array
IP **:** Intellectual Property
LFSR **:** Linear Feedback Shift Register
LUT **:** Look-Up Table
RAM **:** Random Access Memory
RNG **:** Random Number Generator
RO **:** Ring Oscillator
ROM **:** Read Only Memory
RSA **:** Rivest Shamir Adleman
PUF **:** Physical Unclonable Functions

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

FİZİKSEL KLONLANAMAZ FONKSİYONLAR YARDIMIYLA ANAHTAR ÜRETİMİ VE LİSANS DOĞRULAMA

ÖZET

Sahtecilik faaliyetleri global ekonomi için büyük bir tehdittir. Klonlama, istek fazlası üretim, kurcalama (*tampering*), tersine mühendislik başlıca sahtecilik yöntemlerindendir. Lisansız kullanılan veya sahte ürünler birçok firmanın ekonomisine ve marka değerine büyük zararlar vermektedir. Özellikle askeri uygulamalar ile havacılık ve tıp elektroniği uygulamalarında yapılan sahtecilik faaliyetlerinin ölümcül sonuçları olabilmektedir. Sahtecilik faaliyetlerini engellemek için güçlü kriptografik protokollere ve güvenli tanımlama bilgilerine ihtiyaç duyulmaktadır. Fiziksel klonlanamaz fonksiyonlar (PUF) içlerinde çalıştıkları fiziksel ortama özel değerler üretmeleri ve kopyalanamaz olmaları sayesinde güvenli tanımlama bilgilerinin üretilmesinde kullanılabilirler. Bu yapılarla daha güçlü yetkilendirme protokolleri oluşturularak elektronik tasarımların güvenliği arttırılabilir.

Günümüzdeki kriptografik sistemlerin birçoğunda kullanılan anahtarlar ve diğer tanımlama bilgileri bir bellek üzerinde saklanır. Bu gibi kritik bilgilerin sistem içerisinde sürekli varolması zaman zaman güvenlik zaafiyeti oluşturmaktadır. Ayrıca anahtarların üretilmesi ve dağıtılması da sistem güvenliği açısından son derece kritiktir. PUF gibi gerektiği zaman sonucu kolayca okunabilen ve kullanıldığı donanıma özgü çıkış veren fonksiyonlar gerek anahtarın üretilmesi problemine gerekse anahtarın sürekli saklanması problemine karşılık çözüm üretebilirler. Fakat PUF'lar çevresel etkiler ve elektronik gürültü gibi sebeplerle tekrar tekrar okunduklarında çoğu zaman aynı çıkışı veremezler. Kriptografik uygulamalarda kullanılan anahtarlarda hatalı verinin olmaması gerekir. Bu nedenle PUF'lar üzerinde hata giderme işlemi yapılması gerekmektedir. Hata giderme işlemi için sayısal haberleşme sistemlerinde uzun zamandır kullanılan hata düzeltme kodlarının kullanılması (*error correction codes*) önerilmiştir. Hata düzeltme kodları ile hatasız PUF verileri elde edebilmek için elde edilmek istenen hatasız bit sayısından çok daha fazla PUF bitine ihtiyaç duyulmaktadır. PUF ve hata giderme sisteminin kapladığı alanın bunlarla gerçeklenecek sistemlerin geri kalanını baskı altına alacak şekilde büyük olmaması istenmiştir. Bu nedenle seçilecek PUF yapısının az alanda çok sayıda bit sağlaması istenmektedir.

Çalışma kapsamında FPGA platformu üzerinde PUF hataları giderilerek anahtar üretimi gerçeklenmiş ve bu anahtarı kullanan bir lisans doğrulama protokolü oluşturulmuştur. FPGA üzerinde gerçeklemeye uygun olması bakımından halka osilatörlü PUF (RO-PUF) yapısı seçilmiştir. Standart halka osilatörlü PUF yapısı daha az alandan daha fazla bit alabilmek amacıyla yeniden konfigure edilebilir halka osilatörlü PUF yapısına çevrilmiştir. Ayrıca sayıcı çıkışları üzerinde Lehmer ve Gray kodlama yapılmıştır. Hata düzeltme kodu olarak kodlayıcı ve kod çözücüsünün kolay tasarlanabilmesi bakımından birinci derece Reed Muller kodları kullanılmıştır. Elde

edilen hatasız PUF bitleri bir LFSR tabanlı Toeplitz özet fonksiyonundan geçirilerek 128 bitlik anahtar elde edilmiştir. Elde edilen anahtar ile bir lisans doğrulama sistemi (*dongle*) tasarlanmıştır. Buna göre bir IP veya ürün bu lisans doğrulama sistemi ile iletişime geçerek kendisinin lisanslı olup olmadığını kontrol edebilmekte eğer doğru dongle takılı değil ise çalışmamaktadır. Sistemin güvenliği için kritik olan anahtarın dongle dışına çıkmamasını sağlamak üzere bu sistemde RSA açık anahtar kriptografisi tercih edilmiştir. RSA anahtarlarının bazı koşulları sağlaması gerektiğinden PUF çıkışından elde edilen 128 bitlik değer doğrudan anahtar olarak kullanılamamıştır. Bu nedenle PUF çıkışlarından açık ve gizli RSA anahtarlarını üreten bir anahtar üretici blok tasarlanmıştır. Ayrıca RSA şifreleme, şifre çözme, imzalama ve doğrulama işlemlerini gerçekleyen bloklar tasarlanmıştır. Tüm RSA sistemleri yazılımsal olarak gerçeklenmiştir. Platform olarak Microblaze işlemcisi ve büyük sayılarla yapılan işlemler için BigDigits kütüphanesi kullanılmıştır. Çalışma sonunda FPGA platformu üzerinde çalışan PUF tabanlı bir anahtar üretici sistem ve bu sistemi kullanan bir lisans doğrulama sistemi elde edilmiştir.

KEY GENERATION AND LICENSE AUTHENTICATION USING PHYSICAL UNCLONABLE FUNCTIONS

SUMMARY

Counterfeiting products is a serious concern for global economy. Cloning, overproduction, tampering and reverse engineering are some methods of counterfeiting. High tech designs are particularly affected by this threat. Counterfeits of R&D based products are much more damaging than their counterparts. Since there is no R&D expense for counterfeits, producers of these items can make huge profits. Many companies' brand names and economies are damaged by counterfeit or unlicensed products. Counterfeits could also end up being fatal if they are to be used in military, medical and aviation applications. In order to deal with the counterfeiting strong cryptographic algorithms, protocols and secure identifiers are needed.

Physical unclonable functions (PUFs) are functions that create values unique to the physical environments they operate in. They are unclonable due to the impossibility of cloning the physical environment. In semiconductor devices same circuit structures behave different because of the manufacturing process variations. These process variations can not be known prior to characterization of the device. Electronic PUFs exploit these process variations and generates some secret bits from the device. These secret bits can be used as a fingerprint of that device. Tampering with the device that PUF operates in changes the behaviour of the physical device and so changes the PUF responses. Being tamper evident and unclonable, PUFs offer promising security solutions. Such properties of PUFs enables them to be used in cryptographic algorithms. Some examples of PUF structures are SRAM PUFs, ring oscillator PUFs, arbiter PUFs. SRAM PUFs uses the power up behaviour of SRAM cells. Initial state of an SRAM cell is determined by process variations. Ring oscillator PUFs output a response with comparing the nominally identical ring oscillator's oscillation frequencies. Arbiter PUFs uses the race between two paths of same logic structure and determines the winner to give a response. SRAM PUFs and arbiter PUFs are not as suitable to FPGA implementation as ring oscillator PUFs. Because most of the FPGA's initialize the entire chip in a known state at power up SRAM PUFs are not suitable for FPGA implementation. Also symmetry of the paths in arbiter PUFs is crucial and this requirement is very hard to fulfill in FPGAs. In this work mentioned PUFs and many other different PUF structures are examined.

In cryptographic systems generation, distribution and storage of keys are crucial for the security of the systems. Many cryptographic systems store keys and other identification information in memories continuously. This storage mechanism includes many security risks such as probing. Also generation and distribution of the keys brings additional complexity and security risks for the systems. Using PUFs as a key generator is a promising solution against these problems. Identification values can be easily generated when needed without storing the values continuously with PUFs. Also structures of the PUFs are not complex. However PUF responses do not

always generate exactly the same response because of the effects such as noise and temperature. This is a problem for the cryptographic algorithms because even one bit error in a key is not acceptable in these algorithms. An error correction mechanism is needed to correct PUF errors in order to use PUFs as a key generator. There are several methods to correct the errors in a PUF response such as helper data algorithms or fuzzy extractors. In helper data algorithms using error correction codes are offered to correct the PUF errors.

Error correction codes have been widely used in telecommunication applications. They are encoding and decoding mechanisms to transfer messages securely over an unreliable communication channel. Encoding adds redundant bits to the message and decoding corrects the errors and recovers the real message with the help of the redundant bits. In helper data error correction scheme first a random message is chosen and encoded using an error correction code. Then a reference PUF response which is chosen to be the real PUF response is XORed with the generated random code. This data is called an helper data and used for correcting the noisy PUF responses. When a noisy PUF response is generated this response is XORed with the helper data. The result of that operation ends up with the noisy random code. Decoding the code recovers the chosen random message and resulting random code. A final XOR operation of helper data with the recovered code gives the reference PUF response. Using this error correction scheme requires much more PUF bits than generated error free PUF bits used in key generation. Because encoding adds redundant bits to the message but in terms of entropy number of bits are not increased. So area efficient PUF designs are needed in order to supply sufficient amount of PUF bits to the error correction block.

In this work generating error free keys from PUF responses using helper data error correction scheme and using this key in a license authentication system is aimed. All blocks designed in this study are implemented in FPGA platform. Ring oscillator physical unclonable function (RO-PUF) is chosen for its easy implementation in FPGAs. Standard ring oscillator PUF structure is converted to reconfigurable ring oscillator PUF structure to create higher number of bits while occupying smaller area. Also, Lehmer and Gray encoding is applied to the counter outputs to increase the number of bits further. First order Reed Muller codes are chosen as an error correcting code for the helper data key extractor due to its low overhead encoder and decoder structure. Generated error free PUF bits are mapped to a 128 bit key by using LFSR based Toeplitz hash function in order to accumulate the entropy of the entire PUF bits into a 128 bit key.

After generating the error free PUF bits a license authentication protocol is proposed. According to this protocol design of an license authentication dongle is aimed. This license verification dongle can communicate with an IP or a product to confirm its license and if the dongle is not connected to the system it will not funtion. Public key cryptography is chosen for the communication between dongle and the main system. Since in the public key cryptography scheme there is a pair of keys (one is public and the other one is private), there is no problem of key distribution. In this structure the private key does not leave the dongle. But error free PUF responses can not be used as public or private key since these keys have to comply with some mathematical properties. In this work error free PUF responses are used as a seed for the RSA key generator. This secondary key generator and all of the RSA systems are implemented in software. Microblaze processor is used as a platform for the RSA systems. BigDigits software library is used for the operations in very big numbers such as

modular exponentiation, modular multiplicative inverse. All of the encryption, decryption, error correction and PUF blocks are implemented and tested in Spartan 3E family FPGAs. Encryption and decryption systems are implemented in Microblaze platform and rest of the blocks are implemented in hardware. Matlab is used for characterization and testing of the PUF and error correction blocks.

1. GİRİŞ

Ürün sahteciliği günümüz ekonomisini tehdit eden en önemli unsurlardandır. Kopyalama, istek fazlası üretim, tersine mühendislik başlıca sahtecilik yöntemlerindendir [1]. Özellikle elektronik tasarımlarda sahtecilikle yoğun olarak karşılaşılmaktadır. ARGE tabanlı yüksek katma değerli ürünlerde yapılan sahteciliğin zararları çok daha büyük olabilmektedir. Sahte ürünleri yapanlar yüksek yatırım maliyetlerini karşılamadan çok kısa sürede pazara çıkabilme imkanına sahip olduklarından ürünün gerçek sahiplerini büyük zarara uğratmaktadırlar. Sahteciliğin askeri, medikal, havacılık gibi alanlardaki uygulamalarda olmasının sonuçları maddi zararla kalmayıp hayati olabilmektedir. Global olarak satılan yüksek teknolojili ürünlerin %10'unun sahtecilik faaliyetlerinden etkilendiği ve bunun IT sektöründe 100 milyar dolarlık bir gelir kaybı anlamına geldiği tahmin edilmektedir [2].

Üçüncü partilerden alınan veya ürünle beraber geliştirilen IP'ler (*intellectual property*) günümüz tasarım stratejilerinde önemli bir yer tutmaktadırlar. IP'ler yeniden kullanma metodolojisiyle tasarım süresini kısaltırlar. Ayrıca ürünle beraber geliştirilen IP'ler üründen bağımsız olarak satılarak ayrı bir gelir kaynağı olabilmektedir. Sahtecilik tasarlanan ürünlerin tamamı için olduğu gibi IP'ler için de önemli bir tehdittir. Gerek son ürün tasarımlarının gerekse IP tasarımlarının çalışacağı platformlarda gerekli yetkilendirme mekanizmalarını oluşturmak gerekmektedir. Aksi halde kopyalanan tasarımlar kontrolsüz olarak her platformda ve herkes tarafından kullanılabilir olma tehditi altındadır.

Fiziksel klonlanamaz fonksiyonlar (PUF) içlerinde çalıştıkları fiziksel ortama özel değerler üreten ve aynı fiziksel ortamı kopyalamanın imkansız olmasından hareketle kopyalanamaz nitelikteki fonksiyonlardır. PUF'ların bu özelliklerinden faydalanarak tasarım güvenliğini sağlamak üzere yetkilendirme mekanizmaları geliştirilebilir. Tasarım güvenliğini sağlamak üzere kriptografik algoritmaların kullanılması ve bu algoritmalarda kullanılacak anahtarların PUF yardımıyla üretilmesi düşünülmüştür. PUF ile üretilen anahtarlar platforma özel olduğundan tasarımların çalışacağı platformu veya bağlı bulunduğu diğer donanımları doğrulayarak gerekli

yetkilendirmelerin yapılması mümkün olabilmektedir. PUF çıkışlarının üretilmesi kolay olmakla birlikte tekrar tekrar okunduklarında tam olarak aynı sonucu vermediği, gürültü ve sıcaklık gibi etkilerle bazı çıkış bitlerinin değiştiği bilinmektedir. PUF'ların anahtar olarak kullanılabilmesi için PUF bitlerinin sonradan işlenerek hatasız PUF bitlerinin elde edilmesi gerekmektedir. Elde edilen hatasız PUF bitleri o donanıma ait parmak izini oluşturmaktadır. Bu bilgi yetkilendirme uygulamaları için oldukça uygundur.

Günümüzdeki güvenlik uygulamalarının birçoğunda AES gibi simetrik anahtar algoritmaları veya RSA gibi açık anahtar algoritmaları kullanılmaktadır. Bu algoritmalar herkes tarafından bilinmekte ve bu sistemlerin güvenliğini anahtarların gizliliği sağlamaktadır. Bu sistemlerin çoğunda anahtarlar ROM, flash gibi kalıcı (nonvolatile) belleklerde veya harici batarya ile desteklenen kalıcı olmayan RAM'lerde saklanmaktadır. Bu saklama yöntemlerinde anahtarların üretilerek ilgili yere yazılması ve anahtarların kullanılmasa bile sürekli sabit bir yerde saklanması güvenliği azaltmaktadır. Güvenli bir anahtar saklama metodunda şu özellikler bulunur [3] :

- Anahtar sabit bir yerde ve sürekli saklanmamalıdır.
- Anahtar gerekli olduğu zaman donanım içerisinden üretilmeli ve kullanıldıktan sonra bütün iç kaydedicilerden, belleklerden ve diğer yerlerden silinmelidir.
- Anahtar bir şekilde üzerinde bulunduğu donanımla ilişkili olmalı, başka bir donanımda yeniden üretilmesi çok zor olmalıdır.

PUF'lar ile üretilen anahtarlar yukarıda bahsedilen özellikleri sağlayabilirler. Aynı anda hem anahtarın üretilmesi hem de saklanması problemine çözüm üretebilirler ve tasarlanmaları düşük maliyetlidir. Fiziksel atakların uygulanması durumunda verecekleri cevap değişeceğinden bu ataklara karşı korumalıdırlar. Güvenilir yapılabilirler yani hata giderme algoritmaları kullanılarak yüksek doğrulukta hatasız sonuç üretebilirler.

PUF'larda hata gidermek amacıyla haberleşme sistemlerinde yoğun olarak kullanılan hata düzeltme kodları (ECC) kullanılabilir. Bu kodlar ile bulanık düzeltici (*fuzzy extractor*) veya yardımcı veri algoritmaları (*helper data algorithm*) kullanılarak PUF'lar üzerinde hata giderme uygulamaları geliştirilmiştir [4,5,6,7]. Örneğin

[4]'teki çalışmada lineer blok kodlar kullanılarak yardımcı bilgi hata düzeltme yöntemleri çeşitli kodlar için incelenmiştir. Ayrıca sıralanmış kodlar (*concatenated codes*) ve LFSR tabanlı özet fonksiyonu kullanılarak SRAM PUF'larından anahtar üretimi üzerinde durulmuştur. [5]'te halka osilatörlü PUF hataları BCH kodlarıyla düzeltilerek anahtar üretilmektedir. [6]'da blok kodlarda olasılıksal kod çözme kullanılarak SRAM PUF'larında hata düzeltme işlemi yapılmaktadır. [7]'de ise hata düzeltme amacıyla indeks tabanlı sendrom kodlama yöntemi önerilmektedir.

Hata düzeltme algoritmaları şu adımlardan oluşmaktadır:

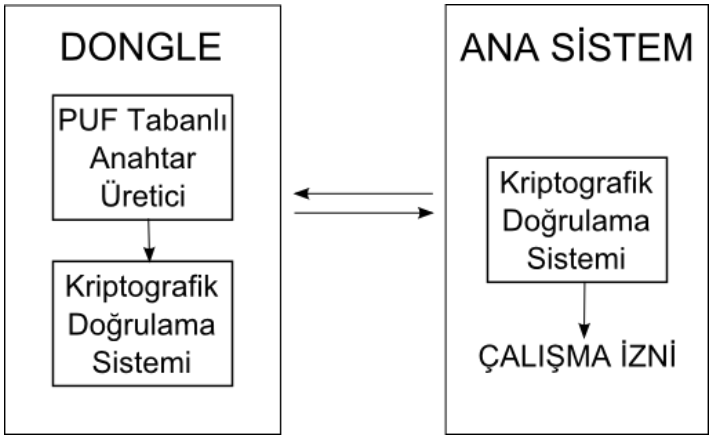
- Kaydolma (*Enrollment*)
- Hata giderme (*Information Reconciliation*)
- Rastgeleliği arttırma (*Privacy Amplification*)

Yardımcı veri algoritmaları hatalı verileri daha önce üretilen yardımcı bilgileri (*helper data*) kullanarak düzeltirler. Yapılan PUF okumalarından biri referans olarak kabul edilir. Bu referans PUF verisi ve hata düzeltme kodları ile yapılan kodlama yardımıyla yardımcı veriler bir defa üretilir ve saklanırlar. Bu adım kaydolma (*enrollment*) adımıdır. Daha sonraki PUF okumalarında oluşan hatalı verileri düzeltmek amacıyla yardımcı veriler ve hata düzeltme kodlarında kod çözme işlemleri kullanılarak hatasız PUF verileri elde edilir. Bu verideki rastgeleliği ve bit dağılımındaki düzgünlüğü (*uniformity*) arttırmak amacıyla hatasız PUF verileri bir özet (*hash*) fonksiyonundan geçirilmektedir.

Elde edilen hatasız PUF bitleri seçilen kriptografik algoritmada ya doğrudan ya da dolaylı olarak anahtar görevi görürler. Eğer simetrik anahtar algoritmaları kullanılacakca PUF verisi doğrudan anahtar olarak kullanılabilir. Eğer açık anahtar algoritmaları kullanılacaksa bu veri özel şartları taşıması gereken anahtarın üretiminde bir başlangıç noktası (*seed*) olarak kullanılabilir. Açık anahtar kriptografisinde gizli anahtarın ve dolayısıyla PUF bilgisinin çip dışına çıkması gerekmeyeceğinden ek alan karşılığında daha fazla güvenlik sağlanır. Bu çalışmada açık anahtar kriptografisi tercih edilmiştir.

1.1 Sistem Mimarisi

Bu çalışmada PUF'lar kullanılarak kriptografik algoritmalarda kullanılmak üzere anahtar üretilmesi amaçlanmıştır. Ayrıca tasarlanan bu anahtar altyapısı ve açık anahtar kriptografisi kullanılarak tasarımlar için bir lisans doğrulama sisteminin oluşturulması amaçlanmıştır. Tüm sistemler sahada programlanabilir kapı dizileri (FPGA) platformu düşünülerek tasarlanmıştır. Tasarlanan sistemin genel yapısı Şekil 1.1'de görülmektedir.



Şekil 1.1: Önerilen lisans doğrulama sistemi.

Sistemin genel yapısında da görüldüğü gibi lisans doğrulama sisteminin tasarımın çalışacağı sistemden ayrı bir donanım olarak tasarlanması (dongle) düşünülmüştür. Buna göre tasarımın yanında kullanıcıya bir dongle verilecek ve tasarımın çalışması için doğru dongle donanımının sisteme bağlı olması gerekecektir. Dongle içerisinde PUF altyapısı kullanılacağından bu donanımın kopyalanması olası değildir. Dongle ile tasarımın çalışacağı platform arasındaki iletişimin üçüncü kişiler tarafından gözlenebilir olduğu durumlarda bu iletişim şifrelenerek yapılmalıdır. Ayrıca kullanılan yetkilendirme protokolünün de yeterince güvenli olması gerekmektedir.

Simetrik anahtar algoritmalarının kullanılması durumunda PUF ile üretilen anahtarın dongle dışına çıkması ve bir şekilde tasarımın çalışacağı sisteme (ana sistem) aktarılması gerekmektedir. Bu durumda aynı anahtarın bir tarafta PUF ile üretilirken diğer tarafta sürekli olarak saklanması gerekmektedir. Bu da PUF'un kullanılma amacı ile çelişen bir durum ortaya çıkarmaktadır.

Bir diğer alternatif ana sistemde bir PUF kullanmak ve dongle donanımına ihtiyaç duymadan tasarımları sadece belirli çiplerde çalışabilir olarak ayarlamaktır. Bu yöntem müşterinin tasarımı çalıştıracağı çiplerin IP tasarımcısının elinde olmasını ve

bu çiplerin PUF cevaplarına göre IP tasarımcısının IP'yi ayarlamasını gerektirmektedir. Müşteriler için bu yöntem pek uygulanabilir olmamaktadır. Sistemin bir IP değil de son ürün olması durumunda ürününün kopyalanmasını engellemek isteyen bir tasarımcının ürünün çalışacağı bütün çiplerdeki PUF cevaplarını incelemesi ve her bir çipe ait özelleşmiş bir tasarım yüklemesi gerekmektedir.

Bu çalışmada önerilen yöntem IP'nin veya son ürünün güvenliğini PUF ve açık anahtar kriptografisi kullanılarak yetkilendirme yapan bir dongle ile sağlamaktır. Açık anahtar kriptografisi ile gizli anahtar dongle dışına çıkmaması, PUF sayesinde dongle donanımının kopyalanamaması ve fiziksel ataklara karşı korumalı olması hedeflenmektedir. Ana sistemdeki tasarım ise açık anahtarı kullanarak dongle ile şifreli iletişim kurmakta ve çalışması için gerekli izinleri elde etmektedir.

Dongle içerisinde bir halka osilatörlü PUF kullanılmıştır. [5]'te gösterildiği gibi halka osilatör ile tetiklenen sayıcıların çıkışları sırasıyla Lehmer ve Gray olarak kodlanarak bit hata oranını arttırmadan elde edilen bit sayısının arttırılması amaçlanmıştır. Halka osilatör hücreleri [8]'de gösterildiği gibi yeniden konfigure edilebilir hücrelere çevrilmiştir. [4]'te bahsedilen yardımcı bilgi algoritması kullanılarak PUF verisi üzerinde hata giderme işlemi yapılmıştır. Kodlayıcı ve kod çözücüsünün kolay tasarlanabilmesi bakımından birinci dereceden Reed Muller kodları tercih edilmiştir. Özet fonksiyonu olarak yine az alan kaplaması bakımından [9]'da bahsedilen LFSR tabanlı özet fonksiyonu kullanılmıştır. Şifreleme sistemi olarak RSA kullanılmıştır. RSA anahtarlarının bazı matematiksel özellikleri sağlaması gerektiğinden PUF özeti kullanılarak anahtar üretme işlemleri gerçekleştirilmiştir. Sistemin bileşenlerine ait detaylar ilerleyen bölümlerde açıklanacaktır. RSA işlemleri Microblaze işlemcisi kullanılarak yazılımsal olarak gerçeklenmiştir. Büyük sayılarla modüler işlemleri yapma kolaylığı sağlayan [10]'da gösterilen "BigDigits" kütüphanesi kullanılmıştır. Tüm sistemler için Spartan 3E FPGA ailesi kullanılmıştır.

2. FİZİKSEL KLONLANAMAZ FONKSİYONLAR

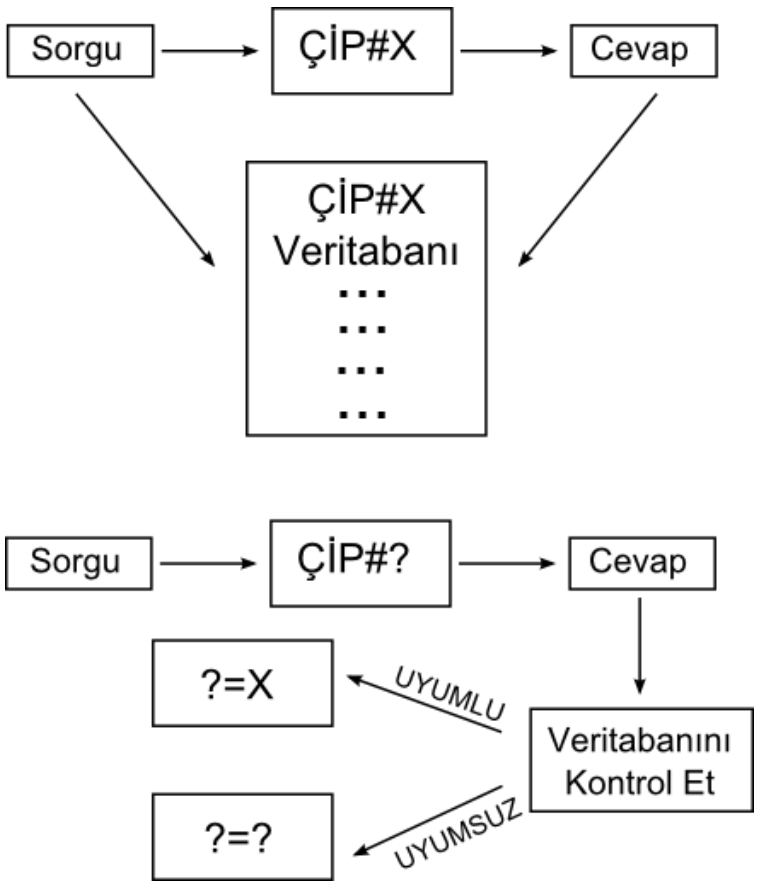
Fiziksel klonlanamaz fonksiyonlar (PUF) üzerinde çalıştıkları fiziksel yapıya özgü bir çıkış üretirler. Bu fiziksel yapıyı modellemek çok zor olduğundan aynı fonksiyon çıkışını başka bir yapı içerisinde elde etmek de olası değildir. Bu nedenle fiziksel klonlanamaz fonksiyonlar ismini almışlardır. Parmak izi gibi biyometrik veriler nasıl sadece bir kişiye aitse, bir PUF verisi de üzerinde çalıştığı fiziksel yapıya aittir. Bu fiziksel yapıya dışarıdan müdahale edilmesi de fiziksel yapıyı bozabileceğinden PUF çıkışını da değiştirebilir. PUF çıkışları gerektiğinde kolaylıkla üretilebilirler. Günümüzdeki bir çok güvenlik uygulamasında tanımlama bilgileri ve anahtarlar bir çeşit bellek üzerinde sürekli olarak saklanmaktadır. Bu bilgilerin sürekli olarak sistem içerisinde varolması zaman zaman güvenlik zaafiyetine yol açmaktadır. PUF çıkışlarının gerektiğinde kolaylıkla oluşturulup okunabilmesi anahtarların ve tanımlama bilgilerininin sürekli saklanmadan gerektiğinde üretilmesi fikrini ortaya çıkarmıştır. PUF'lara ait bahsedilen bu özellikler onların kriptografik uygulamalara bir çok yönden katkı sağlayabileceğini göstermektedir. Literatürde tanımlama, yetkilendirme gibi uygulamalar için PUF'ların kullanılması üzerine bazı çalışmalar bulunmaktadır [11,12]. Bu bölümde öncelikle PUF terminolojisi açıklanacak ve literatürde varolan bazı PUF tiplerinden bahsedilecektir. Daha sonra PUF'ların anahtar üretiminde kullanılması üzerinde durulacaktır. Bu çalışmada kullanılan PUF yapısından bahsedildikten sonra bu PUF'a ait ölçüm sonuçları verilecektir.

2.1 PUF Terminolojisi

PUF'lar bir fonksiyon olduklarından bir giriş kümesine karşılık bir çıkış kümesi üretirler. PUF girişlerine sorgu (*challenge*), PUF çıkışlarına cevap (*response*) ismi verilir. PUF'ları tanımlarken sorgu-cevap çiftleri (*challenge response pair*) kullanılmaktadır.

Bir PUF doğrulanmak istendiğinde daha önceden elde edilmiş sorgu-cevap veritabanıyla sonradan okunan sorgu-cevap çiftleri karşılaştırılmaktadır. Sorgular bazı PUF'larda doğrudan dışarıdan belirlenebilmekle birlikte bazı PUF'larda ise tek

bir sorgu vardır ve bu değer sistem içerisinde belirlenmektedir. Şekil 2.1'de sorgu cevap veritabanı ile doğrulama yöntemi gösterilmektedir.



Şekil 2.1: Sorgu cevap çiftleri ile doğrulama.

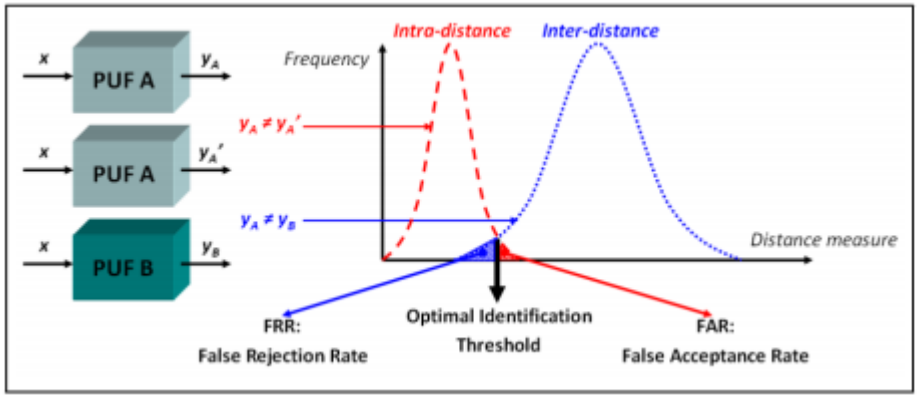
PUF'ları karakterize etmek için iç uzaklık (*intra-distance*) ve ara uzaklık (*inter-distance*) ölçüleri kullanılmaktadır. Bu iki ölçü de Hamming uzaklığı kavramını kullanmaktadır. Hamming uzaklığı eşit uzunluklu iki dizi arasındaki farklı değerli pozisyonların sayısını ifade eder. Bahsedilen diziler karakter dizileri, bit dizileri veya rakam dizileri gibi dizilerdir. Örneğin "110100101" ile "100110101" bit dizileri 2 pozisyonda birbirinden farklı olduğundan bu dizilere ait Hamming uzaklığı 2 olmaktadır. Bu çalışmada Hamming uzaklığı bit dizileri için kullanılacaktır. İç uzaklık aynı sorguya karşılık PUF çıkışlarında elde edilen cevapların birbirine göre Hamming uzaklığını belirten bir ölçüdür. PUF'larda aynı sorguya karşılık verilen cevap tekrar tekrar okunduğunda tamamen aynı cevap çoğu zaman okunamaz. Buna PUF'un bulunduğu yapıdaki gürültü, sıcaklık değişimleri, gerilim değişimleri gibi etkenler sebep olabilir. Tekrar tekrar okunan PUF cevaplarının seçilen bir referans cevaba göre Hamming uzaklıklarının dağılımı o PUF'un iç uzaklık ölçüsünü

belirlemektedir. Bu dağılımın ortalaması PUF'un iç uzaklık ölçüsü olarak tanımlanır. İç Hamming uzaklıklarının toplam bit uzunluğuna oranının 0'a çok yakın olması istenir. Yani bir PUF'u aynı fiziksel yapıda aynı sorgu için tekrar tekrar okuduğumuzda aynı cevapları görmek istenmektedir.

Ara uzaklık ise farklı fiziksel yapılarda çalışan PUF'lara aynı sorgu uygulandığında verdikleri cevapların birbirine göre olan Hamming uzaklığı ile ilgilidir. Bütün sorgu kümesi için bu Hamming uzaklığı çıkarılır. Bu Hamming uzaklıklarının dağılımına ait ortalama PUF'a ait ara uzaklık ölçüsünü verir. Elde edilen Hamming uzaklıklarının bir PUF cevabındaki toplam bit sayısına oranının %50 olması beklenir. Yani bir PUF'a ait sorgu cevap kümeleri eğer biliniyorsa aynı PUF yapısının farklı bir fiziksel yapıda vereceği cevapların aynı olması veya tahmin edilebilmesi istenmez. Tahmin etmenin en zor olduğu durum ise ara uzaklığın %50'ye karşılık geldiği durumdur.

İç uzaklık ve ara uzaklık farklı sorgu cevap çiftlerinde ve farklı PUF çiftlerinde farklılık gösterebilir. Çoğunlukla bu uzaklıklar bir çok sorgu cevap çifti ve PUF için çıkartılarak histogramlar elde edilir. Bu histogramdan elde edilecek eğriler çoğu zaman Gauss dağılımı olarak düşünülebilir. Bu dağılımlara ait ortalama ve standart sapma değerleri belirlenebilir.

Tanımlarına bakıldığında iç uzaklığın PUF'taki okuma hatasına, gürültüye, yeniden üretilebilirliğe ait bir ölçü olduğu, ara uzaklığın ise PUF cevaplarının tekliğine, PUF'ların birbirinden ayırdedilebilirliğine ait bir ölçü olduğu söylenebilir.



Şekil 2.2: PUF'larda iç uzaklık ve ara uzaklık [13].

Şekil 2.2'de iç uzaklık ve ara uzaklık dağılımlarının ayırdedilebilirlik ile ilişkisi gösterilmektedir. İç uzaklık ve ara uzaklık histogramlarının birbiriyle kesişmemesi durumunda herhangi iki PUF'u hatasız olarak birbirinden ayırmak mümkündür.

Fakat Şekil 2.2'deki gibi iki histogramın birbiriyle kesişmesi halinde iki PUF birbirinden yanlış ayrılabilir. Yanlış ayırma olasılığını minumum yapacak Hamming uzaklığı eşiği iki histogramın kesişim noktasıdır.

2.2 PUF Tipleri

Literatürde elektronik ve elektronik olmayan bir çok PUF tipi mevcuttur. Elektronik olmayan PUF tiplerinden bazıları optik PUF, kağıt PUF, CD PUF, RF-DNA, manyetik PUF, akustik PUF olarak sıralanabilir [13]. Burada elektronik olan PUF tiplerine değinilecektir.

2.2.1 V_T PUF

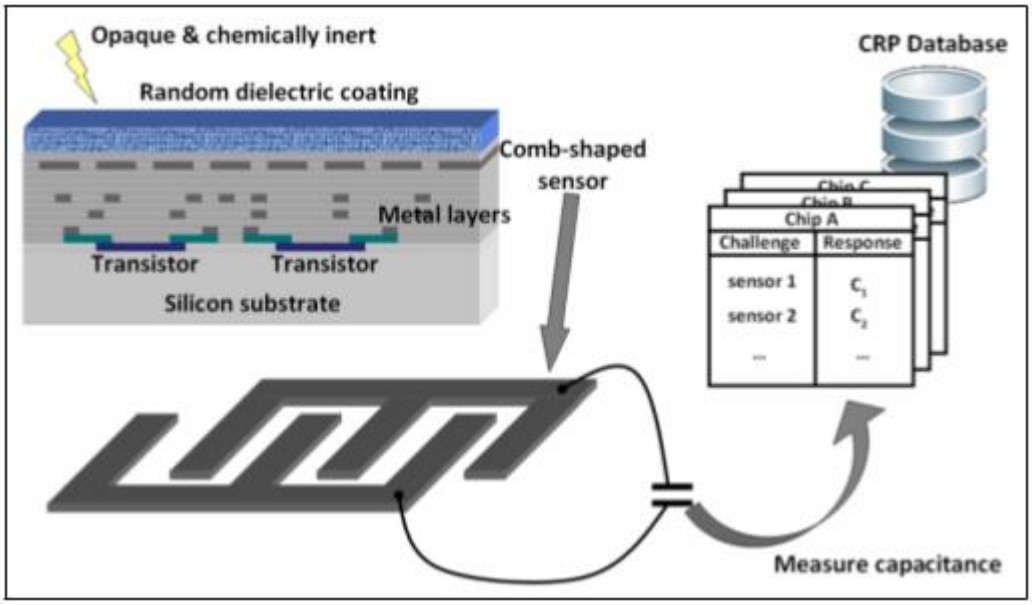
Bir çipte üretilmiş transistörlerin eşik gerilimlerinin birbirinden farklı olması ilkesiyle çalışmaktadır. Aynı şekilde serimi yapılmış transistörler aynı rezistif yükleri sürmekte ve okunan gerilimler karşılaştırıcılar ile bit dizisine dönüştürülmektedir. 0.35 µm CMOS teknolojisinde deneysel bir çalışma yapılmış ve 55 çip üzerinde ortalama iç uzaklık %1.3, ortalama ara uzaklık %50'ye çok yakın olarak elde edilmiştir [13].

2.2.2 Güç dağıtım hattı PUF'u

Bir çipin üzerindeki güç dağıtım hattı dirençlerinin farklılıklarından yola çıkılarak elde edilir. Güç dağıtım hattı üzerindeki gerilim düşümü farklılıkları ölçülerek PUF değerleri elde edilir. 65 nm CMOS teknolojisinde yapılan deneylerde iç uzaklık 0.04 ohm ara uzaklık ise 1.5 ohm olarak elde edilmiştir [13].

2.2.3 Kaplama (Coating) PUF

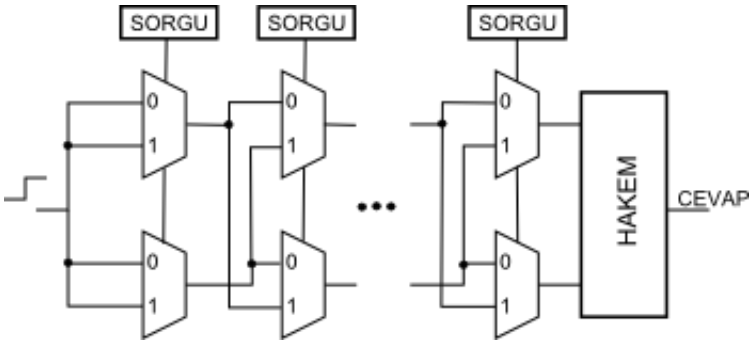
Bir çipin üzerinde tarak şeklinde metal hatların oluşturulması ve dielektrik malzeme ile kaplanması ile yapılırlar. Metal hatların yerleştirilmesindeki ve dielektrik malzemenin yapısındaki üretim farklılıkları nedeniyle metal hatlar arasındaki kapasite değişkenlik göstermektedir. Elde edilen ölçümlerde %50'ye yakın ara uzaklık %5'ten küçük iç uzaklık elde edilmiştir [13]. FIB (*focused ion beam*) gibi işlemler ile çipe müdahele edilmesi durumunda kapasite değerleri değişeceğinden bu PUF kurcalamaya karşı da koruma sağlar (tamper evident). Şekil 2.3'te kaplama (*coating*) PUF yapısı gösterilmiştir.



Şekil 2.3: Kaplama (coating) PUF [13].

2.2.4 Hakem (Arbiter) PUF

Hakem PUF yapısında bir darbe işareti iki farklı yol üzerinden geçer. Yol üzerinde anahtar blokları bulunur. İki farklı hat birbiriyle olabildiğince simetrik tasarlanmalarına rağmen bu hatlardaki geciklemeler üretim farklılıklarından ötürü birbirinden farklıdır. Darbelerden hangisinin önce geldiğine bakarak hakem 0 veya 1 çıkışını vermektedir. Yol üzerindeki anahtar blokları işaretin gittiği yolu ya değiştirmez ya da üstteki işareti alta alttaki işareti üste yönlendirirler. Anahtarın davranışı bir kontrol girişi ile kontrol edilir. Bütün anahtarlara ait kontrol girişleri bir arada PUF'un sorgu (*challenge*) girişlerini oluştururlar. N tane anahtar bulunan bir hakem PUF'ta N bitlik sorgu girişi 1 bitlik cevap çıkışı bulunur. Şekil 2.4'te hakem PUF yapısı göterilmektedir.



Şekil 2.4: Hakem (*arbiter*) PUF.

Bu tip PUF'larda hakem olarak SR-Latch kullanılabilir. Eğer yukarıda bulunan hattan gelen darbe daha geç gelirse SR-Latch'e ait Set girişi Reset girişinden daha

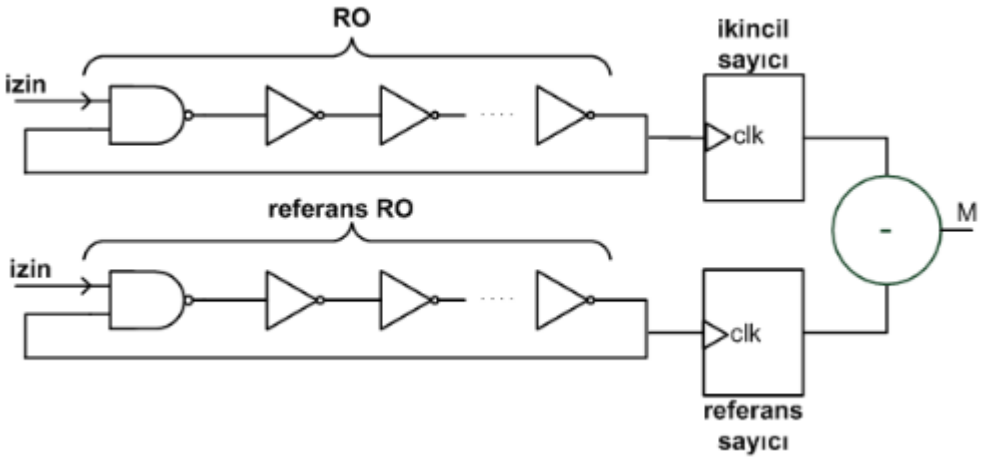
sonra gelecek ve PUF çıkışı 1 olacaktır. Eğer altta bulunan hattan gelen darbe daha geç gelirse SR-Latch'e ait Reset girişi Set girişinden daha sonra gelecek ve PUF çıkışı 0 olacaktır.

Hakem PUF yapılarının mümkün olduğunca simetrik tasarlanması gerekmektedir. Eğer bir hattın nominal gecikmesi diğer hattın nominal gecikmesinden daha büyükse PUF çıkışının bir durumu diğer duruma göre daha olası olmaktadır. Yani PUF çıkışı konumlardan birine daha meyilli olabilmektedir. Oysa olması gereken nominal olarak tamamen eşit gecikmelere sahip hatlardır. Gecikme farklılıklarının sadece üretim farklarından kaynaklanması gerekmektedir.

Hakem PUF yapılarının modelleme ataklarına karşı zayıf olduğu gösterilmiştir [13]. Ayrıca bu yapıyı FPGA'lerde tamamen simetrik tasarlamak routing kaynaklarından gelen kısıtlamalar nedeniyle çok zordur.

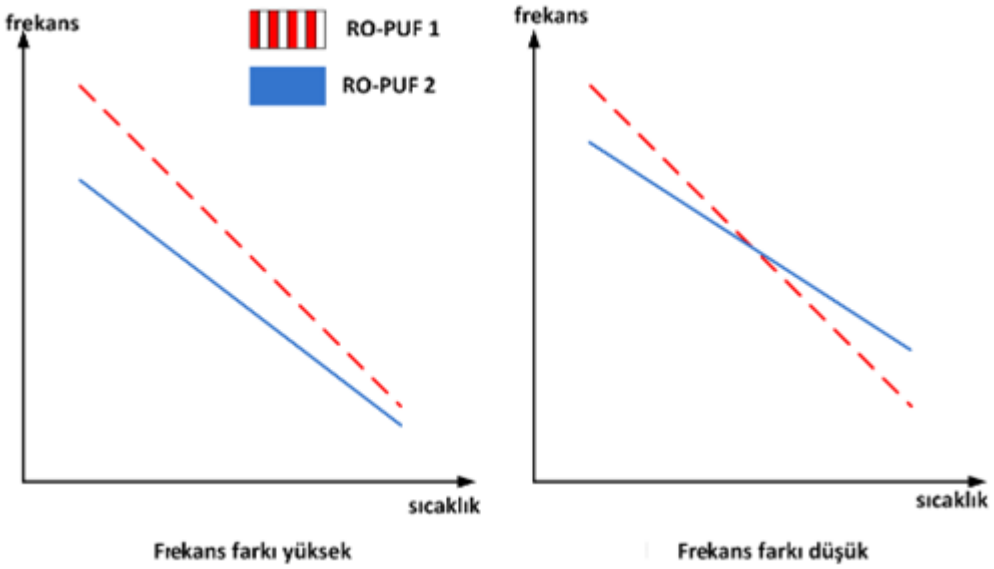
2.2.5 Halka osilatörlü PUF

Halka osilatörler tek sayıda evirme işlemi yapan bloğun ardarda bağlanması ve çıkışın da girişe geribeslenmesi ile oluşturulan osilatörlerdir. Osilasyon frekansını evirme işlemi yapan elemanların toplam gecikmeleri ve hat gecikmeleri belirler. Aynı elemanlardan oluşmuş iki halka osilatörün frekansı üretim farklılıkları sebebiyle birbirinden farklı olacaktır. Halka osilatörünün oluşturduğu darbeler ile sayma yapan sayıcılar ile belirli bir süre sayma işlemi gerçekleştirilir. Sayıcıların çıkışları arasındaki büyüklük küçüklük ilişkisi PUF çıkışını belirler. Ring osilatörlerin sayısı arttırılarak ve bu osilatörlerin sayıcı girişlerine seçilerek bağlanması sağlanarak bu PUF'a sorgu (*challenge*) mekanizması eklenebilir. Bu çalışmada kullanılan PUF halka osilatör tabanlı olup PUF'un yapısı ilerleyen bölümlerde anlatılacaktır. Şekil 2.5'te basit halka osilatörlü PUF yapısı gösterilmektedir. Halka osilatörlü PUF'lar birbiriyle eş gerçeklenebilme bakımından hakem PUF'lara göre FPGA gerçeklemesine daha uygundur. Hakem PUF'larda hat simetresi halka osilatörlü PUF'lara göre çok daha kritiktir. Hat simetrisini FPGA'larda elde etmek ise çok zor olmakla birlikte bazı durumlarda imkansızdır.



Şekil 2.5: Halka osilatörlü PUF.

Halka osilatör frekanslarının birbirine çok yakın olması durumunda çevresel etkenlerin değişimi ile osilatör frekansları arasındaki büyüklük küçüklük ilişkisi değişebilir. Bu da PUF çıkışının tekrar okunması durumunda hatalı sonuç elde edilmesine neden olur. Şekil 2.6'da sıcaklık etkisi ile osilatörler arası frekans ilişkisinin değişimi gösterilmiştir.



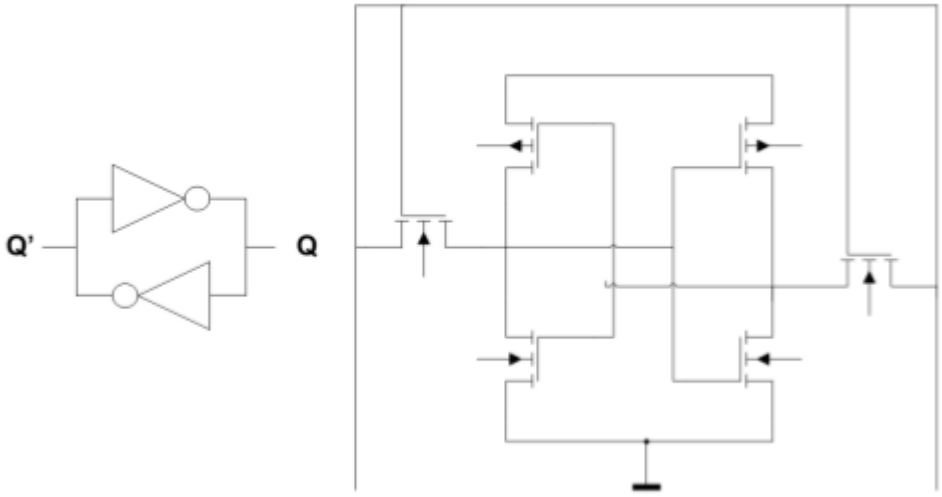
Şekil 2.6: Halka osilatörlerde sıcaklık etkisi [1].

Bu tip hatalar [14]'te gösterildiği gibi etiketleme yöntemi ile düzeltilebilmektedir. Fakat bu yöntemde PUF'a ait istatistiki verilerin bir çok koşulda ve farklı çiplerden elde edilmesi gerekmektedir. Bu bilgiler yardımıyla hata olasılığı en yüksek olan çiftler belirlenir ve etiketlenir. PUF çıkışlarının tekrar okunması sırasında hata olasılığı en yüksek çiftler tekrar değerlendirilir ve PUF çıkışları değiştirilir. Bu hata düzeltme yöntemi yerine eğer N bitlik PUF içerisinde hatalı gelebilecek bitlerin

sayısının belirli bir değerden küçük olacağını kabul edersek bu hatalı bitleri düzeltecek hata düzeltme kodları (*error correction codes*) kullanılabilir. Bu çalışmada hata düzeltme kodları yardımıyla hatasız PUF çıkışları elde edilmiştir. Hata düzeltme kodlarından ilerleyen bölümlerde bahsedilecektir.

2.2.6 SRAM PUF

Şekil 2.7'de görüldüğü gibi bir SRAM hücresi çıkışları çapraz bağlı iki eviriciden oluşmaktadır. Bu eviriciler mümkün olduğunca eş tasarlanır. Farklı hücrelerin ilk güç verildiğinde alacakları çıkış değeri üretim farklılıklarıyla değişebilir. Bu özellikleri nedeniyle SRAM hücreleri PUF olarak kullanılmaktadır.

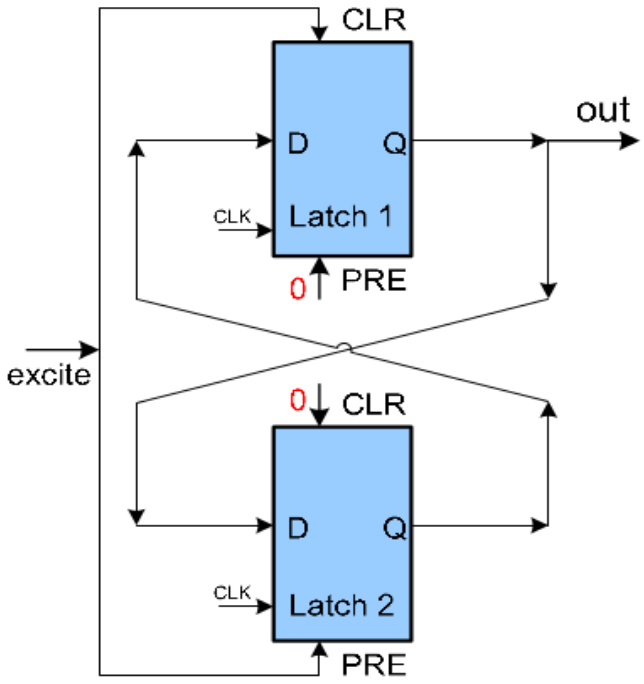


Şekil 2.7: SRAM PUF.

Fakat gerek PUF çıkışlarını her okunmak istendiğimizde gücü kesip yeniden vermenin gerekmesi gerekse FPGA modellerinin birçoğunda ilk güç verilmesi anında tüm birimlerin bir başlangıç değerine zorlanması sebebiyle FPGA gerçeklemesine uygun değillerdir.

2.2.7 Kelebek PUF

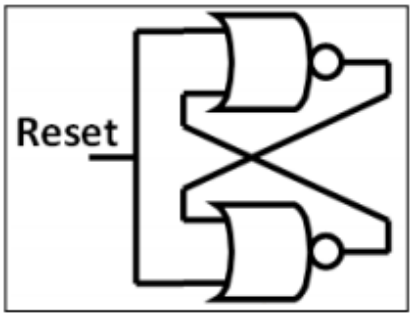
Şekil 2.8'de kelebek PUF yapısı görülmektedir. Bu PUF yapısında çapraz bağlı iki latch kullanılmaktadır. Excite işareti 1 yapıldığında devre kararsız bir konuma gelir. Bir kaç saat işareti sonrasında excite işareti 0 yapıldığında ise mümkün olduğunca simetrik yapılan çapraz bağlama hatlarındaki ve latch elemanlarındaki ufak üretim farklılıkları çıkışın bir değere oturmasını sağlar. Hat simetrisinin çok önemli olması sebebiyle FPGA gerçeklemesine uygun değildir.



Şekil 2.8: Kelebek PUF [15].

2.2.8 Latch PUF

Şekil 2.9'da latch PUF görülmektedir. Kelebek PUF'ta çapraz bağlı iki latch kullanılmıştı, burada çapraz bağlı iki NOR kapısı yani NOR-Latch kullanılmıştır. Reset işaretinin 1 yapılması ile birlikte latch kararsız bir konuma gelir. Reset işaretinin 0 yapılması ile latch çıkışı çoğunlukla üretim farklılıklarının belirlediği bir konuma oturur. 0.13 µm CMOS teknolojisinde yapılan deneysel çalışmada ara uzaklık %50.55, iç uzaklık %3.04 olarak elde edilmiştir [13].



Şekil 2.9: Latch PUF.

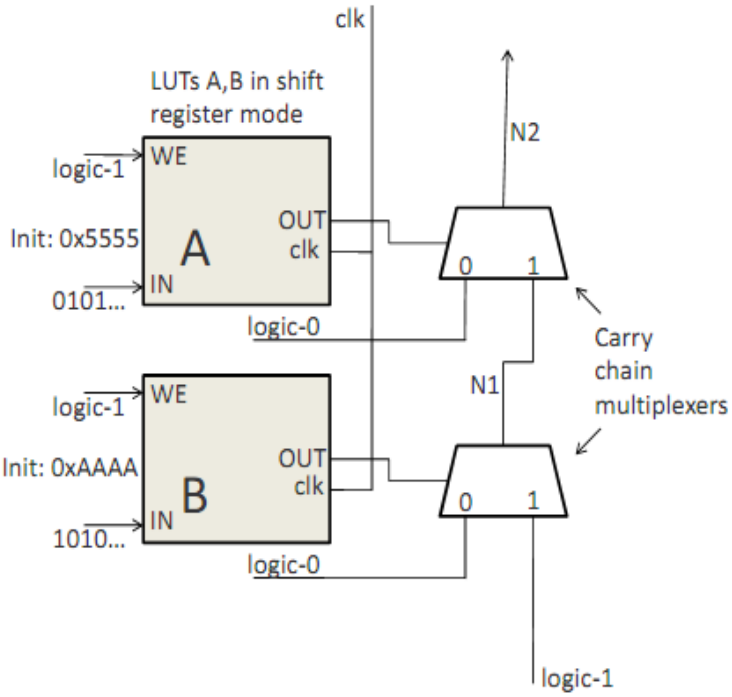
2.2.9 Flip flop PUF

Flip flopların başlangıç gücü verildiğindeki davranışları da SRAM hücrelerine benzer olarak bir PUF olarak kullanılabilir. 3 FPGA'daki 4096 flip flop kullanılarak yapılan çalışmada %11 ara uzaklık %1 iç uzaklık elde dilmiştir. 9'a 1 çoğunluk

oylaması yöntemi ile ara uzaklık %50 civarına çıkarılmış iç uzaklık ise %5'ten küçük kalmıştır [13].

2.2.10 Anderson PUF

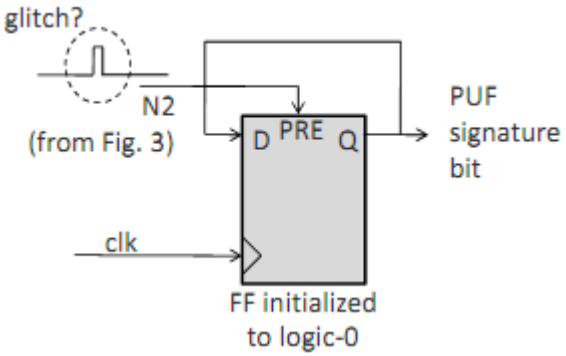
[16]'da önerilen Anderson PUF FPGA mimarisi düşünülerek geliştirilmiştir. Buna göre iki adet LUT'a 0x5555 ve 0xAAAA bit dizileri yüklenmiş ve bu LUT'lar kaydırma moduna alınmıştır. Aynı zamanda kaydırma sonucunda bu bit dizilerinin çıkıştan sürekli okunabilmesi için girişten 0101… ve 1010… girişleri daima verilmektedir. Bu LUT'lar Şekil 2.10'da gösterildiği gibi iki adet çoğullayıcıyı (*multiplexer*) kontrol etmektedir.



Şekil 2.10: Anderson PUF [16].

LUT'lara yüklenen bit dizileri çoğullayıcı kontrollerinin daima birbirinin eviriği olmasını sağlamaktadır. Bu yapıda her saat işareti tetiklemesinin ardından her iki çoğullayıcı konum değiştirmektedir. Eğer üstteki çoğullayıcının kontrol işareti 0, alttaki çoğullayıcının kontrol işareti 1 iken kontrol girişleri konum değiştirdiğinde üstteki çoğullayıcı daha hızlı çıkış verirse bir glitch oluşmaktadır. Bir sonraki konum değiştirmede ise alttaki çoğullayıcı daha hızlı çıkış verirse bit glitch oluşmaktadır. Glitch olması veya olmaması durumu algılanarak buradan 1 bitlik bir cevap elde edilebilir. Bu amaçla Şekil 2.11'de gösterilen glitch algılayıcı kullanılabilir. Virtex 5

modeli bir FPGA'da yapılan deneysel çalışmada 128 bit için ortalama %3.6 iç uzaklık ve %48.3 ara uzaklık elde edilmiştir [16].



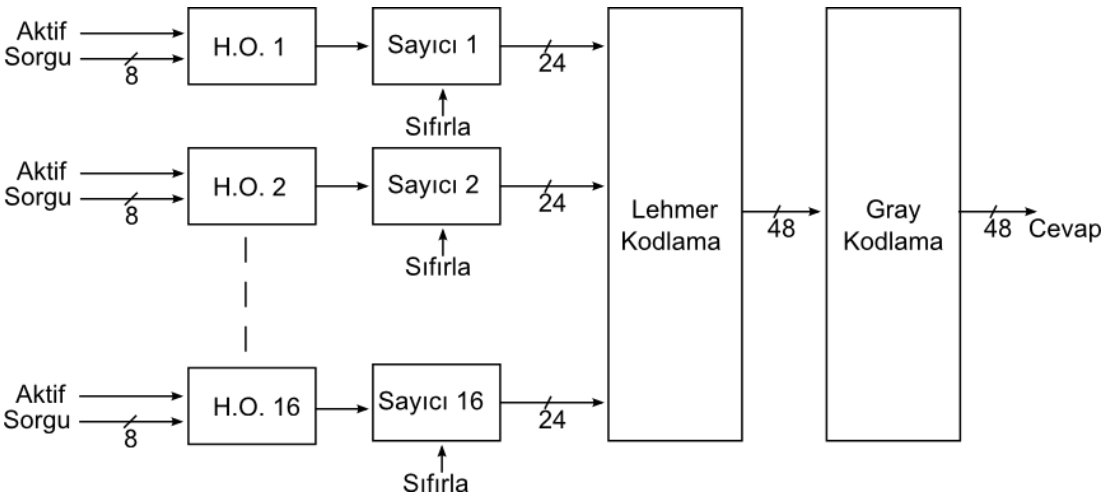
Şekil 2.11: Glitch algılayıcı [16].

2.3 PUF'ların Anahtar Üretiminde Kullanılması

Günümüzdeki birçok güvenlik sisteminde kullanılan anahtarlar ve tanımlama bilgilerinin hem oluşturulması hem de depolanması güvenlik risklerini ve karmaşıklığı beraberinde getirmektedir. PUF'lar daha önce de bahsedildiği gibi üzerinde çalıştıkları fiziksel yapıya özel çıkış ürettiklerinden ve depolamaya ihtiyaç duymadan gerektiğinde kolayca yeniden okunabildiklerinden bu güvenlik bilgilerinin üretilmesi ve depolanması problemine karşı çözümler sunmaktadır. Ayrıca fiziksel yapıya dışarıdan müdahale edilmesi de PUF çıkışlarını değiştirebileceğinden kurcalamaya (*tampering*) karşı da güvenlik sağlayabilirler. Fakat PUF verilerinin tekrar okunduklarında çevresel etkiler, gürültü gibi etkiler sebebiyle tamamen aynı sonucu üretmemeleri anahtar olarak kullanılmaları önünde bir engeldir. Çünkü şifreleme veya şifre çözme gibi kriptografik işlemlerde kullanılan anahtarlarda bir bitin bile farklı olması işlemin başarısız olmasına sebep olmaktadır. Bu engeli aşmak için PUF üzerinde çeşitli hata düzeltme işlemleri uygulanarak hatasız PUF verileri elde etmeye çalışılmaktadır. Bunu yaparken de çipler arası ara uzaklığı (*inter-distance*) %50'ye yakın tutmak yani ayırdediciliği korumak önemlidir. Bu çalışmada hata düzeltme kodları kullanılarak PUF verilerindeki hatalı bitler düzeltilmektedir. Hata düzeltme kodları ile düzeltme yönteminde elde edilmek istenen hatasız PUF bitlerinin sayısından çok daha fazla sayıda PUF bitine ihtiyaç duyulmaktadır. Çünkü seçilen koda göre değişmekle birlikte genel olarak hata düzeltme kodlarında kodun toplam bit sayısı kodlanan mesajın bit sayısından çok daha fazladır. Hata düzeltme kodları bölümünde bu detaylardan bahsedilecektir.

2.4 Kullanılan PUF Yapısı

Bahsedilen PUF tipleri arasında FPGA üzerinde gerçekleme yapmaya en uygunu halka osilatör tabanlı PUF yapısıdır. Bu çalışmada da halka osilatörlü PUF yapısı kullanılmıştır. Hata düzeltme algoritması olarak hata düzeltme kodları (*error correction codes*) kullanılacağından elde edilmek istenen hatasız bitlerin sayısından fazla sayıda PUF bitine ihtiyaç duyulmaktadır. Bu durumda yukarıda bahsedilen halka osilatörlü PUF çok fazla alan kaplayacaktır. Hata düzeltme kodları bölümünde de anlatılacağı üzere toplam 133 hatasız PUF biti elde etmek üzere 1216 PUF bitine ihtiyaç duyulmaktadır. Yukarıda halka osilatörlü PUF tanıtılırken bahsedilen yapının kullanılması halinde 1216 çift yani 2432 adet, referans halka osilatörün bir tane olması durumunda ise 1217 adet halka osilatöre ihtiyaç duyulacaktır. Bu da özellikle FPGA gerçeklemeleri için alan bakımından pratik bir çözüm olmayacaktır. Bu nedenle daha önce bahsedilen halka osilatörlü PUF yapısı daha az alanda daha çok bit elde etmek amacıyla [8]'de bahsedilen konfigure edilebilir halka osilatör yapısı kullanılmış, sayıcı çıkışları da [5]'te bahsedildiği gibi sırasıyla Lehmer ve Gray olarak kodlanmıştır. Kullanılan PUF'a ait blok şema Şekil 2.12'de görülmektedir.



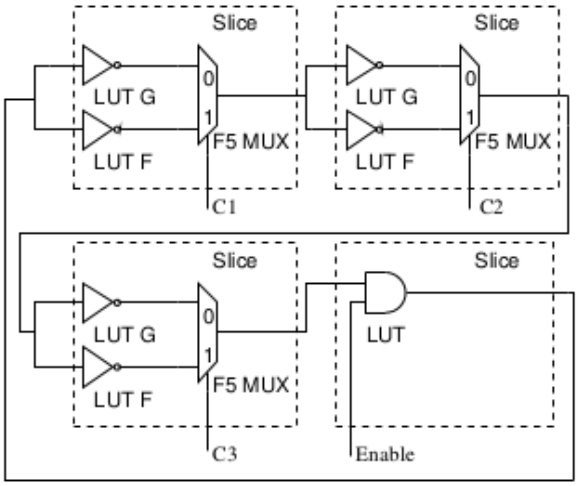
Şekil 2.12: Kullanılan PUF yapısı.

Bu çalışmada 8 bitlik sorgu girişi ile 256 farklı osilatör oluşturabilecek yeniden konfigure edilebilir halka osilatörden 16 adet paralel olarak kullanılmıştır. Her bir paralel osilatör ile tetiklenen 24 bitlik 16 adet sayıcı bulunmaktadır. 16 adet 24 bitlik değer sırasıyla Lehmer ve Gray olarak kodlanmakta ve her bir sorgu için 48 bitlik PUF verisi elde edilmektedir. 256 farklı sorgu için bu yapı ile toplam 12288 PUF biti elde etmek mümkündür.

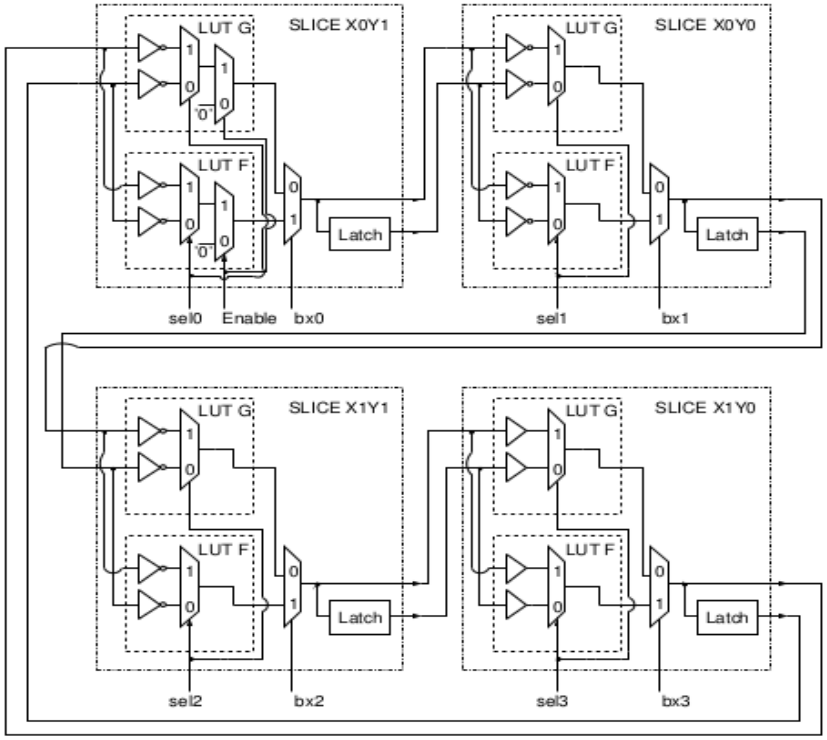
Kullanılan Spartan 3E FPGA üzerinden 16 adet halka osilatör 16 CLB, 16 adet 24 bitlik sayıcı 48 CLB, Lehmer ve Gray kodlama blokları 72 CLB olmak üzere PUF tasarımı için toplam 136 CLB kullanılmıştır.

2.4.1 Yeniden konfigure edilebilir halka osilatör

[8]'deki ve [17]'deki çalışmalarda önerilen halka osilatör yapıları FPGA gerçeklemeleri göz önünde bulundurularak önerilmiş ve 1 CLB içerisinde maksimum halka osilatör konfigurasyonu elde etmek amaçlanmıştır. [17]'deki çalışmada 1 CLB alan kaplayan ve 3 bitlik sorgu girişleri ile 8 farklı halka osilatör konfigurasyonu elde edilebilen bir halka osilatör yapısı önerilmiştir. Şekil 2.13'te bu osilatör yapısı görülmektedir. [8]'de ise [17]'deki yapı daha da geliştirilmiş ve bir CLB içerisinde 256 farklı halka osilatör konfigürasyonu elde edilmesine imkan sağlanmıştır. Şekil 2.14'te bu osilatör yapısı görülmektedir. Bu halka osilatörlerde öncekilerin aksine aynı elemanlar kullanılarak farklı osilatörler elde edilebilir. Bu da daha küçük alan ile daha çok PUF biti elde edilmesine olanak tanır. Bu çalışmada [8]'de önerilen halka osilatör kullanılmıştır.

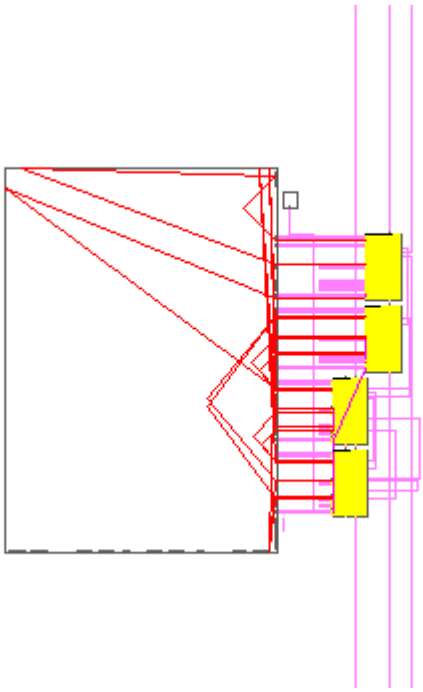


Şekil 2.13: 3 bitlik sorgu girişli yeniden konfigure edilebilir halka osilatör.



Şekil 2.14: 8 bitlik sorgu girişli yeniden konfigure edilebilir halka osilatör.

Kullanılan 16 halka osilatör hücresinin birbirinin lojik blok olarak olduğu gibi ara bağlantılarda (*routing*) da tamamen eş olabilmesi için bir konfigure edilebilir osilatör hücresi tasarlanmış ve bu hücre donanımsal makro (*hard macro, blackbox*) haline getirilmiştir. Spartan 3E modeli FPGA'lar için bir CLB alan kaplayan makro FPGA Editor adlı yazılım ile elde edilmiştir. Bu makronun görünümü Şekil 2.15'te görülmektedir. Gerçeklenen halka osilatörü hücreleri ve çıkış sayıcıları Spartan 3E FPGA üzerinde toplam 201 slice, 384 flip flop ve 384 LUT kullanmaktadır.



Şekil 2.15: Makro olarak tanımlanmış bir halka osilatör hücresi.

2.5 Lehmer ve Gray Kodlama

Halka osilatörler ile yapılan PUF'larda PUF bitlerine karar vermenin bir yolu sayıcı çıkışlarını çiftler halinde birbiriyle karşılaştırmaktır. Bu durumda iki adet halka osilatörden 1 bitlik çıkış elde edilir. Lehmer kodlama ise bir permutasyon kodlama yöntemidir. N adet değerin permutasyonlarının büyüklük küçüklük sıralamasının sayısal olarak ifade edilmesi olarak da ifade edilebilir. Bu kodlamayla çiftler halinde yapılan karşılaştırmaya göre daha fazla bit elde edilebilir.

i∈ (1,2..N) olmak üzere **N** adet $\mathbf{C_i}$ sayıcı değeri için oluşturulan **N-1** adet Lehmer katsayısı $\mathbf{L=(L_1,L_2,...L_{N-1})}$ şu şekilde tanımlanmaktadır:

$$L_j = \sum_{i=1}^{j} gt(C_j + 1, C_i) \tag{2.1}$$

$$gt(x,y) = \begin{cases} 1 & x > y \\ 0 & x \leq y \end{cases} \tag{2.2}$$

Tanıma göre her bir Lehmer katsayısı $\mathbf{L_i}$, i değerinden büyük olmamak üzere negatif olmayan tamsayı değerlerini alabilir. Bu durumda Lehmer katsayıları vektörü **L 2x3x...xN=N!** farklı değer alabilir. Bu da N adet değerin permutasyonlarının sayısına

eşit olduğundan Lehmer kodlamanın bir permutasyon kodlama yöntemi olduğu söyleneblir.

Bu çalışmada 16 adet (N=16) paralel sayıcı girişi Lehmer olarak kodlanmış ve her adımda 15 katsayı elde edilmiştir.

Lehmer katsayılarının önemli bir özelliği komşu iki değerin yer değiştirmesi ilgili Lehmer katsayısını sadece ∓ 1 değiştirmektedir. Bu değişimlerin sadece 1 bitlik bir değişmeye yol açması için Lehmer katsayıları Gray olarak kodlanmıştır. Gray kodlamanın özelliği ardışık sayılar arasında sadece 1 bitlik farkın olmasıdır. PUF verisinin tekrar tekrar okunması sırasında oluşacak bit hatalarının her bir frekans ilişkisi değişiminde 1 bitlik olabilmesi için Gray kodlamanın kullanılması gerekmektedir. Lehmer katsayıları $\mathbf{L_i}$'nin 0'dan i'ye kadar değer alabileceğinden bahsedilmişti. Bir Lehmer katsayısının entropi değeri $\log_2(i+1)$ olarak hesaplanır. Bu durumda N adet giriş olması durumunda elde edilen toplam entropi ise (2.3)'teki gibi hesaplanır.

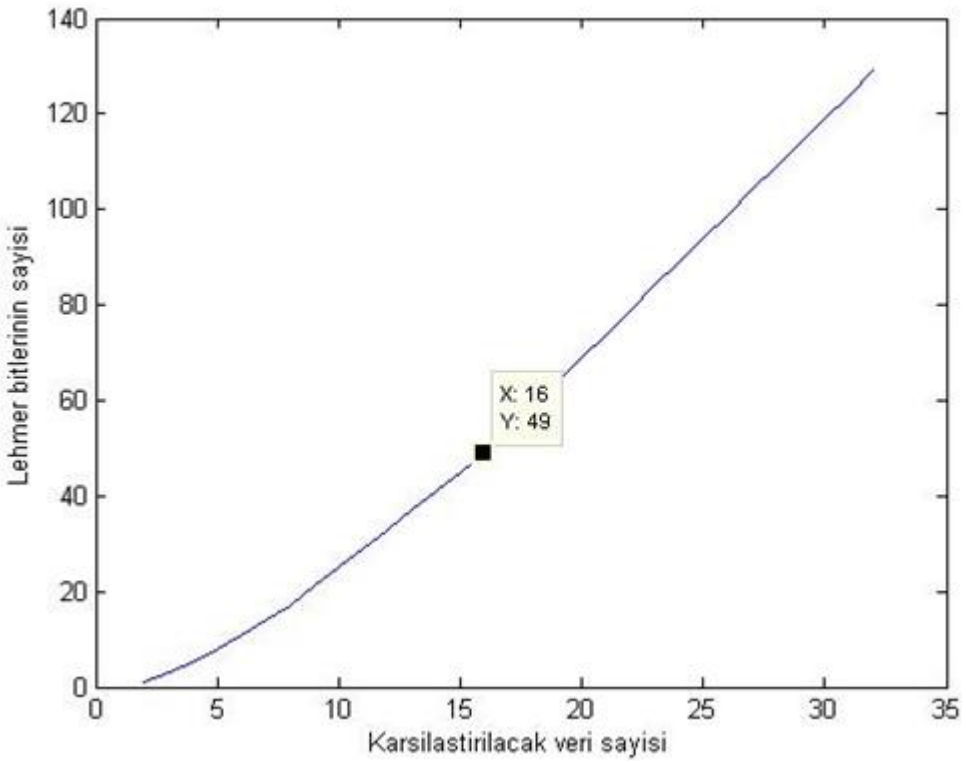
$$H = \sum_{i=2}^{N} \log_2 i \qquad \textbf{(2.3)}$$

Gray kodlama ile her bir Lehmer katsayısı ile elde edilen bit sayısı ise ilgili entropinin yukarıya yuvarlanmış halidir. Buna göre Gray kodlama sonucu elde edilen toplam bit sayısı

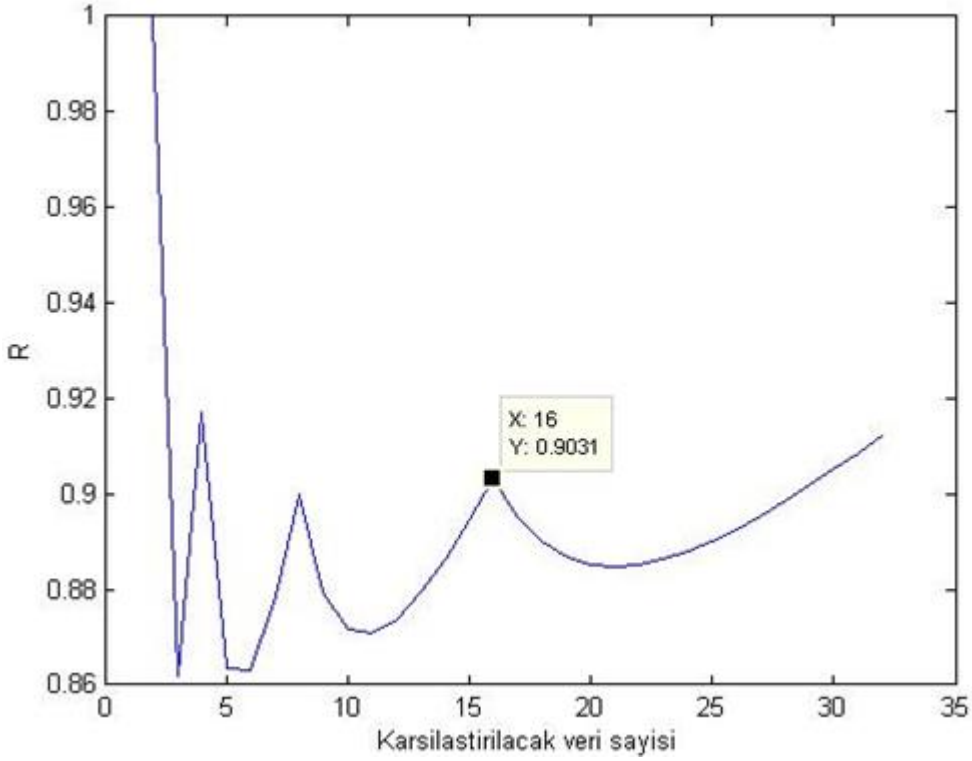
$$H' = \sum_{i=2}^{N} \lceil \log_2 i \rceil \qquad \textbf{(2.4)}$$

şeklinde hesaplanır.

H ile H' değerinin oranına entropi oranı (R) denir. Entropi oranı 1'den küçük olması bazı bitlerin çoğunlukla 0 veya çoğunlukla 1 olduğu anlamına gelir. Bu da PUF'lar arası uzaklığı azaltıcı bir durumdur ve ayırdedilebilirliği azaltır. Bu nedenle öncelikle entropi oranı ve çıkıştaki bit sayısı ile Lehmer kodlayıcı girişlerinin sayısının değişimi ve incelenmiştir. Bu değişimler Şekil 2.16 ve Şekil 2.17'de gösterilmiştir.



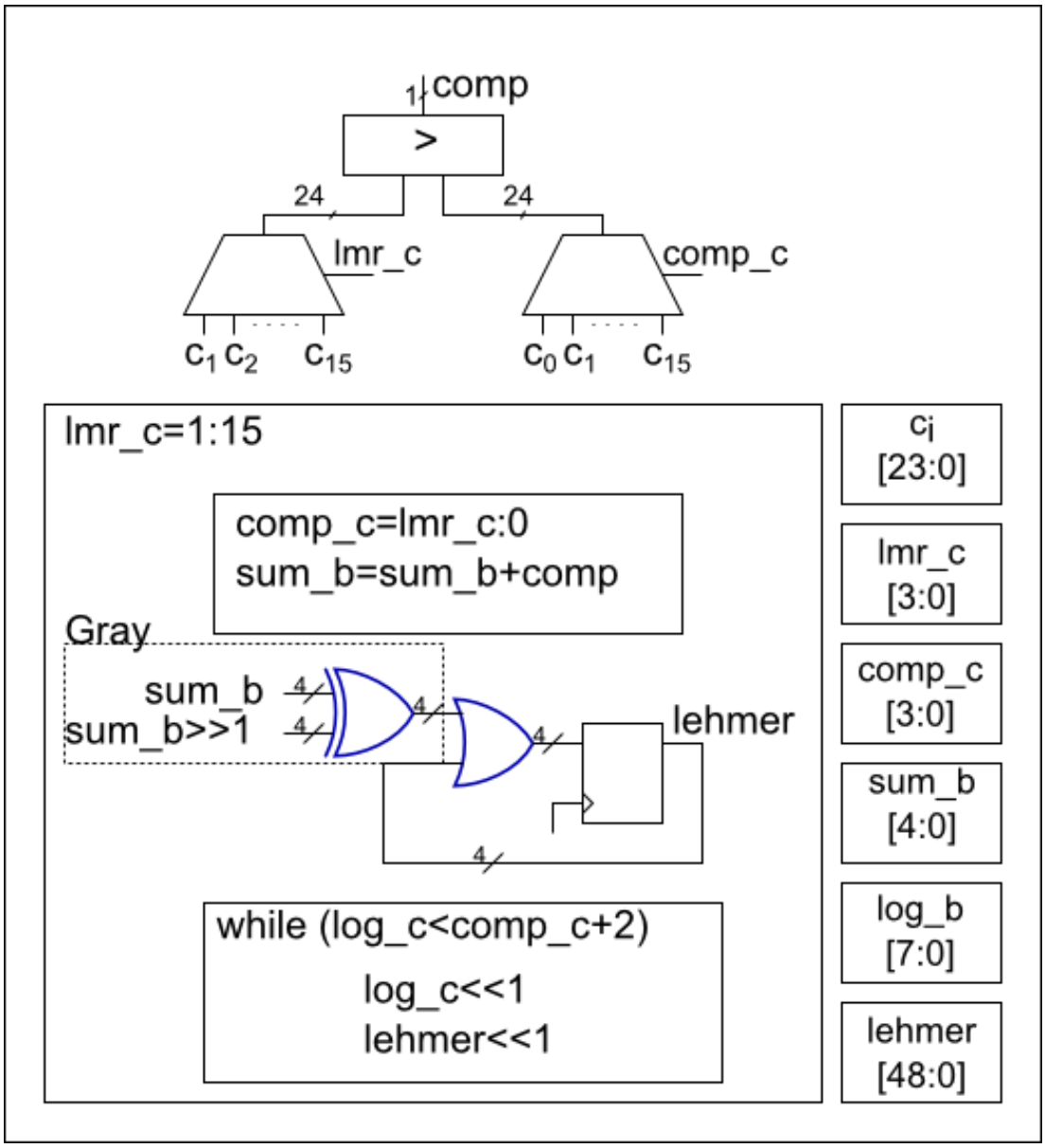
Şekil 2.16: Çıkıştaki bit sayısı ile karşılaştırma yapılan veri sayısının değişimi.



Şekil 2.17: Entropi oranının Lehmer kodlayıcı girişlerinin sayısı ile değişimi.

Buna göre 16 paralel halka osilatör hücresi ve 16 sayıcı kullanıldığında Lehmer ve Gray kodlama sonucunda elde edilen bitlerin sayısı 49 olmakta ve bu durumda entropi oranı 0.9031 olmaktadır. 49 bitten çoğunlukla sabit kalan bitlerden biri başka bir bitle XOR işleminden geçirilmiştir. Böylelikle çıkıştan 48 bit alınmakta ve entropi oranı 0.9219 olmaktadır. Bu işlem ile bit sayısı düşürülerek entropi oranı arttırılabilmekte fakat işlem sonucu elde edilen bitteki hata olasılığı artmaktadır. Bu çalışmada 16 paralel halka osilatör kullanılarak her iki adımda alınan 96 bitten 64'ü hata düzeltme kodunda kullanılarak toplam 38 adımda 1216 bit elde edilmiştir.

Çalışmada gerçeklenen Lehmer ve Gray kodlama bloğunun mimarisi Şekil 2.18'de gösterilmiştir. Buna göre C_i kodlama bloğunun sayıcı girişlerini göstermektedir. Kodlama bloğunun içerisinde de lmr_c ve comp_c sayıcıları bulunmaktadır. Bu iki sayıcı ile hangi girişlerin birbiriyle karşılaştırılacağı seçilmektedir. Karşılaştırma sonuçları toplanarak sum_b kaydedicisinde tutulmaktadır. Bu değer Gray olarak kodlanmakta ve sonuç ilgili Lehmer katsayısını vermektedir. Her bir Lehmer katsayısı 49 bitlik lehmer kaydedicisine aktarılmaktadır. Bunu yaparken Lehmer katsayılarının kaç bit olacağı log_b kaydedicisi yardımıyla belirlenmekte ve lehmer kaydedicisinin belirlenen bit sayısı kadarlık kısmına sum_b sonucu aktarılmaktadır. Lehmer ve Gray kodlama bloğu Şekil 2.18'de gösterilen mimariye uygun olacak şekilde Verilog donanım tanımlama dili ile gerçeklenmiştir. Spartan 3E FPGA üzerinde toplam 285 slice, 95 flip flop, 544 LUT kullanılmıştır.

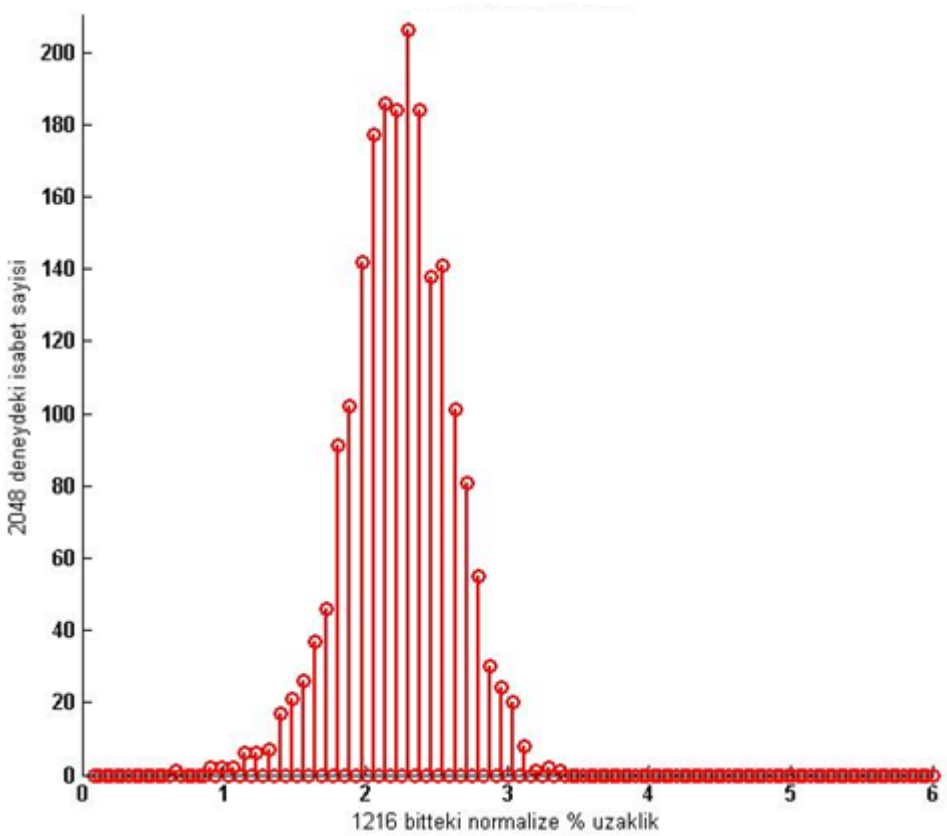


Şekil 2.18: Lehmer ve Gray kodlayıcının gerçeklenmesi.

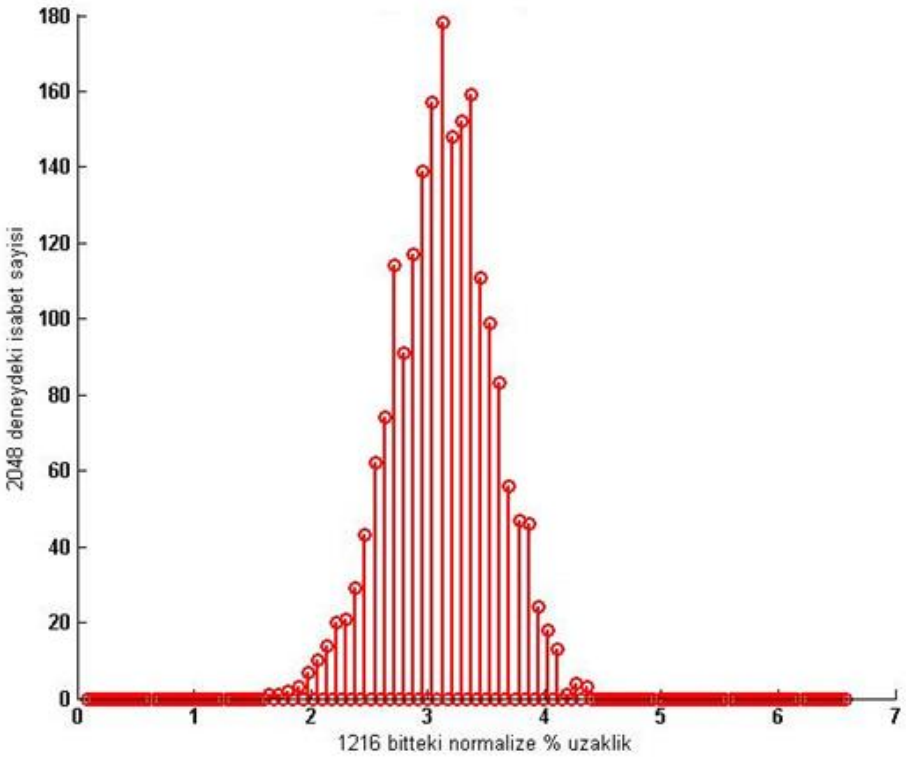
2.6 Gerçeklenen PUF'a Yapısının Testi

Gerçeklenen halka osilatörlü PUF'un incelenmesi için ayrı bir test sistemi oluşturulmuştur. Bu test sisteminde Microblaze işlemcisi, PUF ve UART modülü bulunmaktadır. Bilgisayar üzerinden Microblaze işlemcisine UART yardımıyla komut gönderilmektedir. Microblaze işlemcisi 48 bit ve 16 bitlik adımlarla toplam 32 adımda 1216 PUF bitini almakta ve bunları yine UART üzerinden bilgisayara göndermektedir. İki farklı FPGA'da test sistemi koşturulmuş, PUF verileri 2048 defa alınmış ve bilgisayar ortamına aktarılmıştır. Alınan PUF verileri Matlab üzerinde Lehmer ve Gray olarak kodlanmıştır. Gray kodlanmış PUF verileri üzerinde analizler

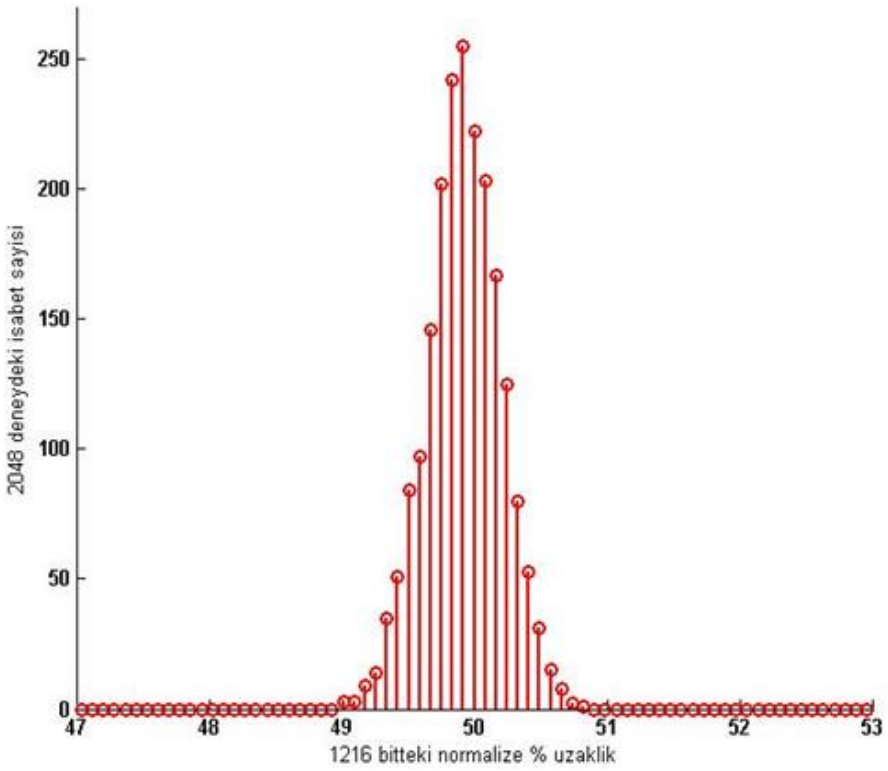
yapılmıştır. Aynı FPGA'lara ait veriler üzerinde iç uzaklık dağılımı, farklı FPGA'lara ait veriler üzerinde ara uzaklık dağılımı çıkarılmıştır. Sonuçlar Şekil 2.19, Şekil 2.20 ve Şekil 2.21'de gösterilmektedir.



Şekil 2.19: 1. FPGA'da çalışan PUF'a ait iç uzaklık dağılımı.



Şekil 2.20: 2. FPGA'da çalışan PUF'a ait iç uzaklık dağılımı.



Şekil 2.21: 1. ve 2. FPGA'larda çalışan PUF'lar arası uzaklığın dağılımı.

Elde edilen sonuçlara ait ortalama ve standart sapma değerleri Çizelge 2.1'de gösterilmiştir.

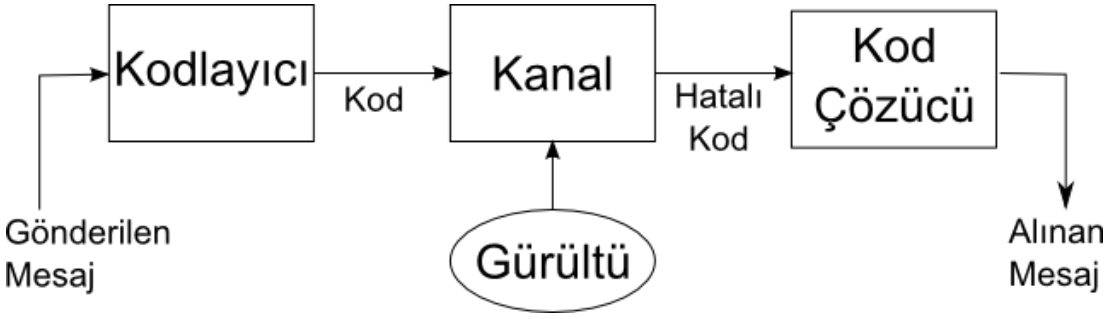
Çizelge 2.1 : Alınan PUF verilerine ait istatistikler.

Analiz	Ortalama	Std. Sapma
İç Uzaklık 1	%3.13	%0.43
İç Uzaklık 2	%2.24	%0.36
Ara Uzaklık 1-2	%49.92	%0.28

Burada verilen uzaklık dağılımları yapılan deney sayısı arttırılarak yani daha fazla sayıda PUF verisi elde ederek Gauss dağılımına yakınlaştırılabilir.

3. HATA DÜZELTME KODLARI (ECC)

Haberleşme uygulamalarında kanal kodlama gürültülü bir kanaldan yapılan haberleşmedeki hataları kontrol edebilmek için kullanılan bir yöntemdir. Bu yöntemdeki temel fikir gönderilen veride mesaj verisinin yanında hataların belirlenmesinde ve düzeltilmesinde yardımcı olacak ekstra verinin de bulunmasıdır. Gönderilen verideki bu fazlalık belirli bir sayıya kadar hataları belirlemeye imkan vermektedir. Mesajları kodlamak, yani mesaj bitlerinin yanındaki yardımcı veriyi oluşturmak üzere birçok kodlama yöntemi önerilmiştir. Bu kodlara tekrar (Repetition), BCH, Reed-Solomon, Reed-Muller ve Hamming kodları örnek olarak verilebilir. Kodlama işlemi ikili (*binary*) veya ikili olmayan (*nonbinary*) sayılar üzerinde blok olarak veya konvolüsyonel olarak yapılabilmektedir. Blok kodlar içerisinde ikili olmayan kodlara Reed-Solomon kodları, ikili olan kodlara Hamming kodları örnek olarak verilebilir. Bu çalışmada ikili lineer blok kodlar üzerinde durulacaktır.



Şekil 3.1 : Haberleşme kanalında hata giderimi.

3.1 İkili Lineer Blok Kodlar

Mesaj uzunluğu k olan ve kod uzunluğu n olan bir ikili (*binary*) lineer blok kod $\boldsymbol{\mathcal{C}}$, $\mathbb{F}_2^n$ alanının **k** boyutlu bir alt uzayıdır. Kod kelimeleri mesajların $\mathbb{F}_2^n$ üzerindeki karşılıklarıdır. İki farklı **u** ve **v** kodu $\mathbf{u}=(u_1,u_2,\ldots,u_n)$ ve $\mathbf{v}=(v_1,v_2,\ldots,v_n)$ vektörleri ile ifade edilir. Burada $u_i,v_i \in \mathbb{F}_2$ koşulu sağlanır. Mesaj bitlerinin sayısı **k** ise 2^k farklı mesaj oluşturulabilir. Bu mesajların kodlanması ile 2^k farklı **n** bitlik kod kelimesi elde edilebilir. Dolayısıyla **n** bitlik verilerle oluşturulabilecek bütün kelimelerin

sayısı $\mathbf{2^n}$ olup bu kelimelerin hepsi bir koda karşılık gelmez. Bunlardan $\mathbf{2^k}$ tanesi geçerli bir koddur. Lineer kodlarda geçerli kodların lineer birleşimi de geçerli bir koddur.

Kodlar arası Hamming uzaklığı iki kod kelimesi karşılaştırıldığında birbiriyle farklı olan bitlerin sayısı olarak tanımlanmıştır. Minimum uzaklık da bir kodu tanımlayan parametrelerden biri olup $\mathbf{2^k}$ adet geçerli kodların birbirleriyle olan Hamming uzaklıklarının minimumudur. Bir lineer blok kodun uzunluğu **n**, mesajın uzunluğu **k** ve kod kümesine ait minumum uzaklık **d** kullanılarak **(n,k,d)** olarak gösterilir. $\boldsymbol{\frac{k}{n}}$ oranına kod oranı denir. Bu oranın düşük olması kodlama sonucunda daha fazla fazlalık (*redundant*) bitlerin ortaya çıkması anlamına gelir. Lineer blok kodların hataları düzeltebilmesini sağlayan özellik minimum uzaklıktır. Hata düzeltme işlemi temelde kod çözerken gelen verinin geçerli kodlardan hangisine Hamming uzaklığı bakımından en yakın olduğunun belirlenmesi işlemidir. Buradan yola çıkarak bir lineer blok kodunun hata düzeltme kapasitesi $\mathbf{t=\lfloor\frac{d-1}{2}\rfloor}$ bit olarak tanımlanır. Yani alınan bir **n** bitlik veriden hatasız **n** bitlik kodun kesin olarak elde edilebilmesi için bu **n** bit içerisinde bulunan hataların sayısı maksimum **t** bit olmalıdır.

Bir lineer kod üretici matrisi (*generator matrix*) ile de tanımlanır. Üretici matrisinin satırları lineer bir kod için bazdır. Bir üretici matris için geçerli lineer kodların tamamı üretici matrisin satırlarının tüm lineer birleşimleridir. Yani lineer kod, üretici matrisin satır uzayıdır. Bu durumda bir mesaja karşılık gelen lineer kod şu şekilde ifade edilir.

$$\vec{C} = \vec{m}.[G]_{kxn} \qquad \textbf{(3.1)}$$

G matrisi üretici matris olup **k**x**n** boyutundadır. **G** matrisinin rankı **k**'dır. Mesaj vektörü $\boldsymbol{m}$, **k** boyutlu ve code vektörü **C** ise **n** boyutludur. Bir mesaj kodlanmak istendiğinde ilgili kodun üretici matrisi ile çarpılmaktadır.

G üretici matrisi satır işlemleri ve sutun değiştirme işlemleri ile $\mathbf{[I_k \mid P]}$ formuna dönüştürülebilir. Buradaki **P** matrisi kullanılarak $\mathbf{H=[P^T \mid I_{n-k}]}$ matrisi tanımlanır. Bu matrise parite kontrol matrisi (*parity check matrix*) adı verilir. Bu matrisin boyutu **(n-k)xn** olmaktadır. Bir lineer kodun geçerli bir kod olabilmesi için şu ilişkiyi sağlaması gerekir:

$$\vec{\mathbf{0}} = [\boldsymbol{H}]_{(\boldsymbol{n-k})\boldsymbol{xn}}.\vec{\boldsymbol{C}} \qquad \textbf{(3.2)}$$

Yani bir geçerli kodlardan oluşmuş alt uzay, üretici matrisin satır uzayı olduğu gibi parite kontrol matrisinin de sıfır uzayıdır. Parite kontrol matrisi aynı zamanda **(n,k,d)** olarak tanımlanan **C** kodunun duali olan **(n,n-k,d')** kodunun üretici matrisidir. Minumum uzaklık ise parite kontrol matrisinin linear bağımlı sutunlarının minimum sayısıdır. Yani minimum uzaklık **d** ise parite kontrol matrisinde **d-1** sütundan oluşan bütün kümelerde bu sütunlar lineer bağımsız olup, **d** sütunlu ve birbiriyle lineer bağımlı olan bir sütun kümesi bulunur.

Kod çözme yöntemlerinden biri minimum uzaklık kod çözme (*minumum distance decoding*) yöntemidir. Bütün kod kümesine ait kodların bir tablo halinde elimizde bulunduğunu düşünelim. Gelen veri ile bütün kodlar arasındaki uzaklık tek tek hesaplanarak minimum uzaklığı sağlayan kod seçilebilir. Fakat çok büyük kod tabloları için tek tek karşılaştırma yapmak pratik bir uygulama değildir. Bunun için bu kod tablolarının boyutlarını küçülterek kod çözme yapan sendrom kod çözme yöntemi önerilmiştir. Gelen hatalı veri **z=c+e** şeklinde ifade edilebilir. Burada **c** geçerli bir kod kelimesi, **e** ise hata vektörüdür. Daha önceden **H.c=0** olduğu gösterilmişti. **H.z** değerini hesaplarsak **H(c+e)=0+H.e=H.e** olduğunu görürürüz. **H.z=H.e** değerine sendrom ismi verilmektedir. Bir lineer kod maksimum $\mathbf{t=\left\lfloor\frac{d-1}{2}\right\rfloor}$ adet hata içeren verileri düzeltebileceğinden gerçekleşebilecek e vektörlerinin sayısı tüm geçerli kod kelimelerinin sayısından küçüktür. Dolayısıyla minimum uzaklığı veren kod kelimesi aranırken tüm kod kelimelerini tek tek aramaktansa sendrom ifadesini sağlayan **e** vektörünü belirlemek daha pratik bir çözüm olacaktır. Sendrom ifadesini sağlayan **e** vektörünü bulmak için denenmesi gereken **e** vektörlerinin sayısı

$$\sum_{i=0}^{t} \binom{n}{i} < |C| \qquad \textbf{(3.3)}$$

olarak ifade edilir. Doğru **e** vektörü bulunduktan sonra gerçek kod kelimesi **c=z-e** olarak kolayca hesaplanabilir.

İkili kodlarda bütün matris işlemleri modülo 2 olarak gerçekleştirilir. Bu durumda matris çarpımında kullanılan çarpma işlemi lojik "AND" işlemine toplama işlemi ise lojik "XOR" işlemine karşılık gelmektedir.

3.2 Reed-Muller Kodları

Reed-Muller kodları Irving S. Reed ve David E. Muller tarafından bulunmuştur. Muller kod yapısını önermiş Reed ise çoğunluk lojiği ile kod çözme yöntemini bulmuştur. 1972 yılında Mariner 9 uydusu tarafından, Mars'ın siyah-beyaz fotoğraflarını gönderebilmek için kullanılmıştır [18]. İlk olarak ikili (*binary*) olarak önerilmişse de diğer tabanlara ait genelleştirmeleri de bulunmaktadır.

Reed-Muller kodları RM(r,m) şeklinde gösterilirler. Kodun derecesini r, uzunluğunu m belirler. Klasik gösterimde kullanılan n,k ve d parametreleri şu şekilde hesaplanır:

$$n = 2^m \tag{3.4}$$

$$k = \sum_{i=0}^{r} \binom{m}{i} \tag{3.5}$$

$$d = 2^{m-r} \tag{3.6}$$

Bu çalışmada kodlayıcı ve kod çözücüsünün kolay tasarlanması ve az alan kaplaması sebebiyle 1. derece Reed-Muller kodları kullanılmıştır. 1. derece Reed-Muller kodları klasik kod gösterimi ile **(n,k,d)=(2^m,m+1,2^{m-1})** şeklinde gösterilir. Bu kodlarla **t=**$\left\lfloor \frac{d-1}{2} \right\rfloor = \frac{n}{4} - 1$ adet hatalı bit düzeltilebilir.

Birinci derece Reed-Muller kodlarında kodlama ve kod çözme işlemlerine geçmeden önce ikili düzende vektörler üzerindeki toplama, çarpma ve iç çarpım işlemlerinden bahsedilecektir. Daha büyük dereceli Reed-Muller kodları için [18]'deki çalışma incelenebilir.

x ve **y** vektörleri ile a skaleri $\mathbb{F}_2 = \{0, 1\}$ üzerinde tanımlı vektörler olup toplama işlemi **x+y=**(x_1+y_1,x_2+y_2,…x_n+y_n), **a+x**=(a+x_1,a+x_2,…,a+x_n) olarak tanımlanır. Çarpım işlemi **x*y=**(x_1*y_1,x_2*y_2,…x_n*y_n), **a*x**=(a*x_1,a*x_2,…,a*x_n) olarak iç çarpım işlemi ise **x.y=**x_1*y_1+x_2*y_2+…+x_n*y_n olarak tanımlanır. Burada + işlemi "XOR" işlemine * işlemi ise "AND" işlemine karşılık gelir.

Birinci derece Reed-Muller kodu RM(1,m) için üretici matris (m+1)x2^m boyutludur. Matrisin satırları sırasıyla her biri n elemanlı **1**, $\mathbf{x_1}$,$\mathbf{x_2}$,…$\mathbf{x_m}$ vektörlerinden oluşur. **1** vektörü 2^m adet 1'den oluşur. Diğer $\mathbf{x_i}$ vektörleri ise her biri toplam n adet eleman

içerecek şekilde 2^{m-i} adet 1 ve 2^{m-i} adet 0 sırasını takip eden elemanlara sahiptir. Örneğin RM(1,3) koduna ait üretici matris şu şekildedir:

$$\begin{bmatrix} \mathbf{1} \\ \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \boldsymbol{x_3} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad \textbf{(3.7)}$$

Reed-Muller kodlarında kodlama işlemi diğer kodlarda olduğu gibi üretici matris ile çarpma işlemi yapılarak yapılmaktadır. Kod çözme işlemi ise Reed tarafından önerilen çoğunluk lojiği kod çözme yöntemi (*majority logic decoding*) ile yapılır. Birinci derece kod için kod çözme işleminde şu adımlar izlenir:

- Üretici matristeki her bir $\mathbf{x_i}$ vektörü için 2^{m-1} adet karakteristik vektör bulunur. Karakteristik vektörlerin bulunması örnek ile gösterilecektir.
- Kodu çözülecek veri ile her satırın karakteristik vektörlerinin iç çarpımı hesaplanır. Her bir satırda hesaplanan iç çarpım sonuçlarının çoğunluk değeri o satıra karşılık gelen mesaj bitini (m_i) verir.
- Üretici matrisin birinci satırına karşılık gelen mesaj bitini bulmak için ise $\boldsymbol{My} = \sum_{i=1}^{m} \boldsymbol{x_i} * \boldsymbol{m_i}$ vektörü hesaplanır. Bu vektörün başlangıçtaki kodu çözülecek vektör ile toplamının elemanlarının çoğunluk değeri üretici matrisin birinci satırına karşılık gelen doğru mesaj bitini vermektedir.

Örnek olarak RM(1,3)=(8,4,4) kodundaki kod çözme işlemini ele alalım. Bu kod 1 adet hatalı biti düzeltme kapasitesine sahiptir. **m**=(0,1,1,0) mesajı bu kod ile kodlandığında $\mathbf{M_c}$=(0,0,1,1,1,1,0,0) kod kelimesi elde edilir. Kodun sisteme bir bit hatalı $\mathbf{M_e}$=(1,0,1,1,1,1,0,0) verisi olarak geldiğini kabul edelim. Sırasıyla $\mathbf{x_i}$ vektörlerinin karakteristik vektörleri bulunmalıdır.

- $\mathbf{x_3}$ vektörüne ait karakteristik vektörler $x_1 * x_2$, $x_1 * \overline{x_2}$, $\overline{x_1} * x_2$, $\overline{x_1} * \overline{x_2}$
- $\mathbf{x_2}$ vektörüne ait karakteristik vektörler $x_1 * x_3$, $x_1 * \overline{x_3}$, $\overline{x_1} * x_3$, $\overline{x_1} * \overline{x_3}$
- $\mathbf{x_1}$ vektörüne ait karakteristik vektörler $x_2 * x_3$, $x_2 * \overline{x_3}$, $\overline{x_2} * x_3$, $\overline{x_2} * \overline{x_3}$

olarak verilmektedir. $\mathbf{x_3}$ vektörüne ait karakteristik vektörlerin $\mathbf{M_e}$ ile iç çarpımları

(11000000).(10111100)=1, (00110000) .(10111100)=0,

(00001100) .(10111100)=0, (00000011) .(10111100)=0.

olarak bulunur. Bu değerlerin çoğunluğu 0 olduğu için başlangıçtaki mesaj vektörü **m**'nin 4. elemanı m_4 0'dır.

$\mathbf{x_2}$ vektörüne ait karakteristik vektörlerin $\mathbf{M_e}$ ile iç çarpımları

(10100000).(10111100)=0, (01010000) .(10111100)=1,

(00001010) .(10111100)=1, (00000101) .(10111100)=1.

olarak bulunur. Bu değerlerin çoğunluğu 1 olduğu için başlangıçtaki mesaj vektörü **m**'nin 3. elemanı m_3 1'dır.

$\mathbf{x_1}$ vektörüne ait karakteristik vektörlerin $\mathbf{M_e}$ ile iç çarpımları

(10001000).(10111100)=0, (00100010) .(10111100)=1,

(01000100).(10111100)=1, (00010001) .(10111100)=1.

olarak bulunur. Bu değerlerin çoğunluğu 1 olduğu için başlangıçtaki mesaj vektörü **m**'nin 2. elemanı m_2 1'dır.

m'nin 1. elemanını bulmak üzere $\mathbf{M_y}=m_4*\mathbf{x_3}+ m_3*\mathbf{x_3}+ m_2*\mathbf{x_3}$ vektörü hesaplanır ve bu vektör $\mathbf{M_e}$ vektörü ile toplanır.

$\mathbf{M_y}$=0*(10101010)+1*(11001100)+1*(11110000)=(00111100)

olarak bulunur.

$\mathbf{M_y}+\mathbf{M_e}$=(00111100)+(10111100)=(10000000)

vektörünün elemanlarının çoğunluğu 0 olduğundan mesaj verisinin birinci biti de 0 değerini alır. Böylelikle doğru mesajımız **m**=(0110) olarak elde edilmiştir.

3.3 Yardımcı Veri Algoritması ile PUF Hatalarının Düzeltilmesi

PUF verisinde hata düzeltme işlemi iki aşamadan oluşur. Birinci aşama referans PUF verisinin okunması ve yardımcı verinin (*helper data*) oluşturularak belleğe kaydedilmesi işlemidir. Bu işlemin bir defa tekrarlanması yeterlidir. Bu işlemden "Gen" işlemi olarak bahsedilecektir. İkinci aşamada ise PUF verisi tekrar okunduğunda oluşan hataların yardımcı veri yardımıyla düzeltilmesi işlemidir. Hata düzeltme işlemi sonucunda referans PUF verisi yeniden elde edilmiş olur. Bu işlemden "Rep" işlemi olarak bahsedilecektir.

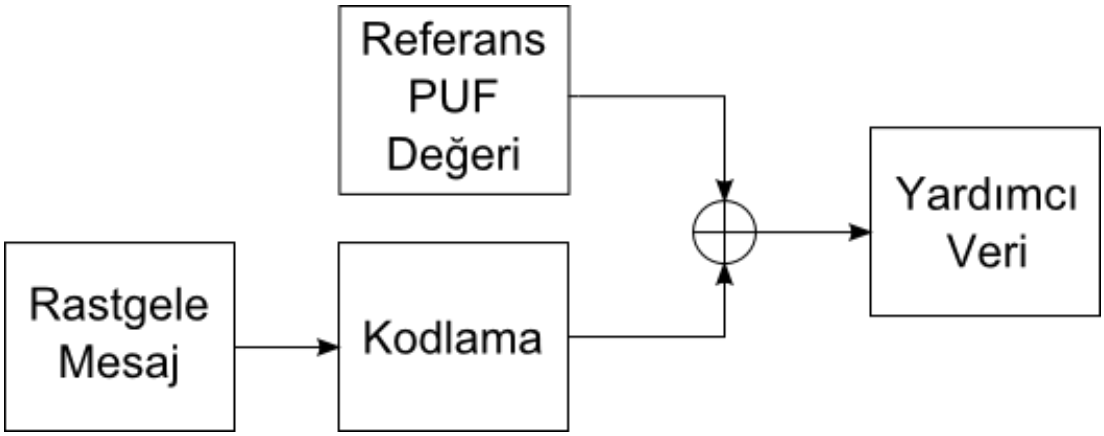
Gen işlemi şu adımlardan oluşur:

- Elde edilmek istenen hatasız PUF bitlerinin sayısı $\mathbf{s_k}$ belirlenir.
- (n,k,d) hata düzeltme kodu seçilir. (k<=$\mathbf{s_{k)}}$
- Toplam adım sayısı $\lceil \frac{s_k}{k} \rceil$ olarak, $\mathbf{s_k}$ adet hatasız bit elde etmek için gereken PUF bitlerinin sayısı ise $n.\lceil \frac{s_k}{k} \rceil$ olarak hesaplanır.
- Her bir adımda k bitli rastgele bir mesaj seçilir. Bu mesaj kodlanarak n bitlik rastgele kod elde edilir. Aynı zamanda n adet PUF biti okunur. Rastgele kod ile PUF bitleri XOR işleminden geçirilir. Elde edilen sonuç ilgili adıma ait yardımcı bilgi olarak kaydedilir. Bu işlem $\lceil \frac{s_k}{k} \rceil$ adım için tekrar edilir.

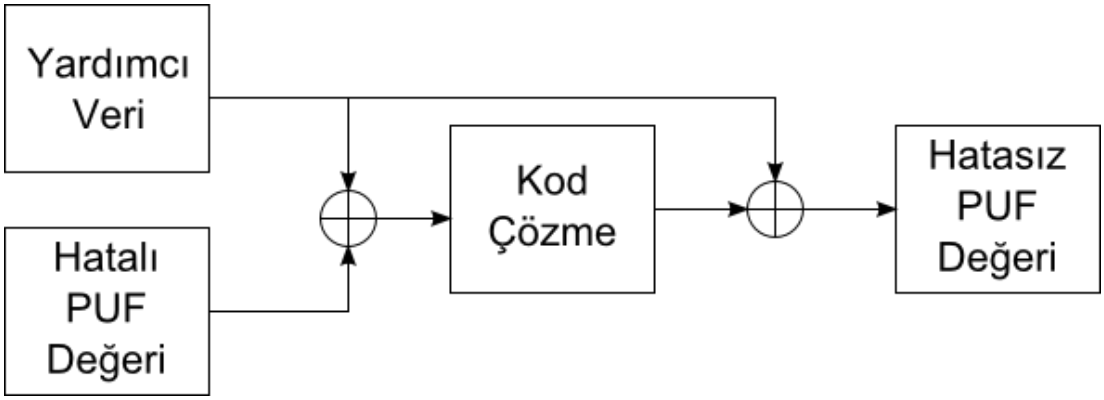
Rep işlemi ise şu adımlardan oluşur.

- Bir adıma ait yardımcı veri okunur.
- Yardımcı veri oluşturulurken okunan n PUF verisi tekrar okunur. Bu veride hata bulunması beklenmektedir.
- n bitlik PUF verisi ile n bitlik yardımcı veri XOR işleminden geçirilir. Elde edilen veri yardımcı veri elde edilirken üretilen rastgele kod değerinin hatalı bir versiyonudur.
- Elde edilen hatalı kod için (n,k,d) koduna ait kod çözme işlemi yapılarak gerçek kod değeri elde edilir.
- Gerçek kod değeri ile yardımcı veri XOR işleminden geçirilerek ilgili adıma ait n bitlik gerçek PUF değeri elde edilir. Bu verinin k biti hatasız PUF verisi olarak kullanılır.
- Bu işlem $\lceil \frac{s_k}{k} \rceil$ adım için tekrarlanarak $k.\lceil \frac{s_k}{k} \rceil$ adet hatasız PUF verisi elde edilir.

Rep işlemindeki her adımda n adet hatasız PUF verisinin elde edilmesine rağmen bunun k adet bitinin kullanılmasının sebebi bir (n,k,d) kodunun n bitlik olmasına rağmen k bitlik entropi içermesidir. Çünkü n bitlik bir kodda oluşturulabilecek kodların sayısını mesaj bitlerinin sayısı k belirler. Mesaj bitlerinin sayısı k olan bir kodda 2^k adet n bitlik kod oluşturulabilir. Gen ve Rep işlemleri Şekil 3.2 ve Şekil 3.3 ’te gösterilmiştir.

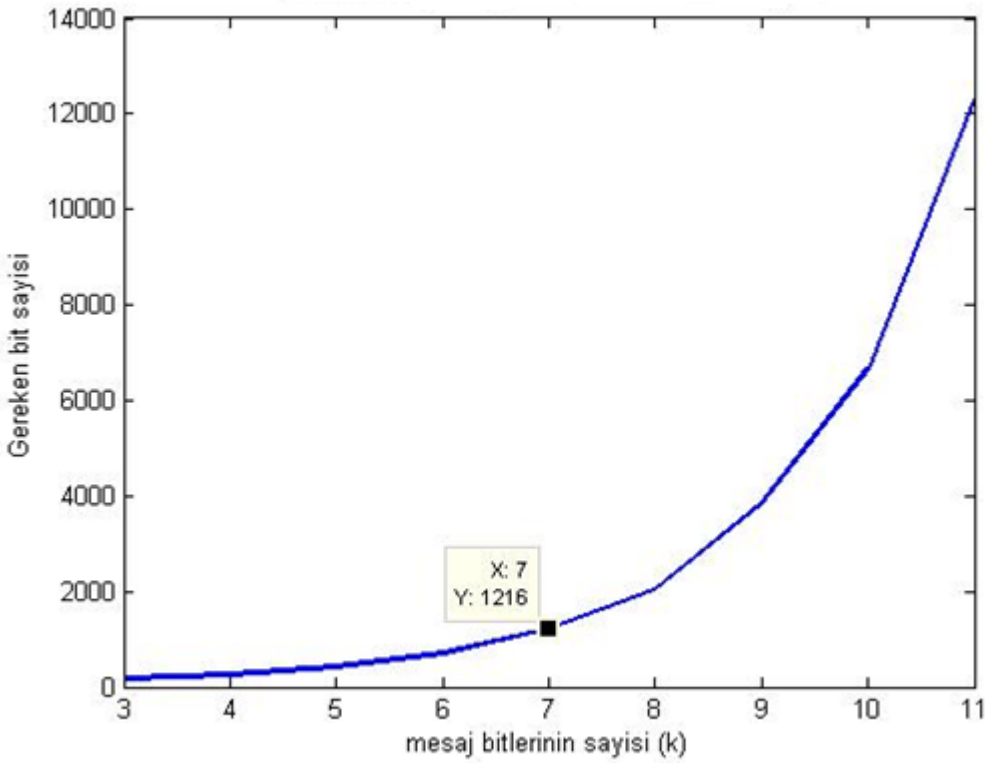


Şekil 3.2 : Gen işlemi.

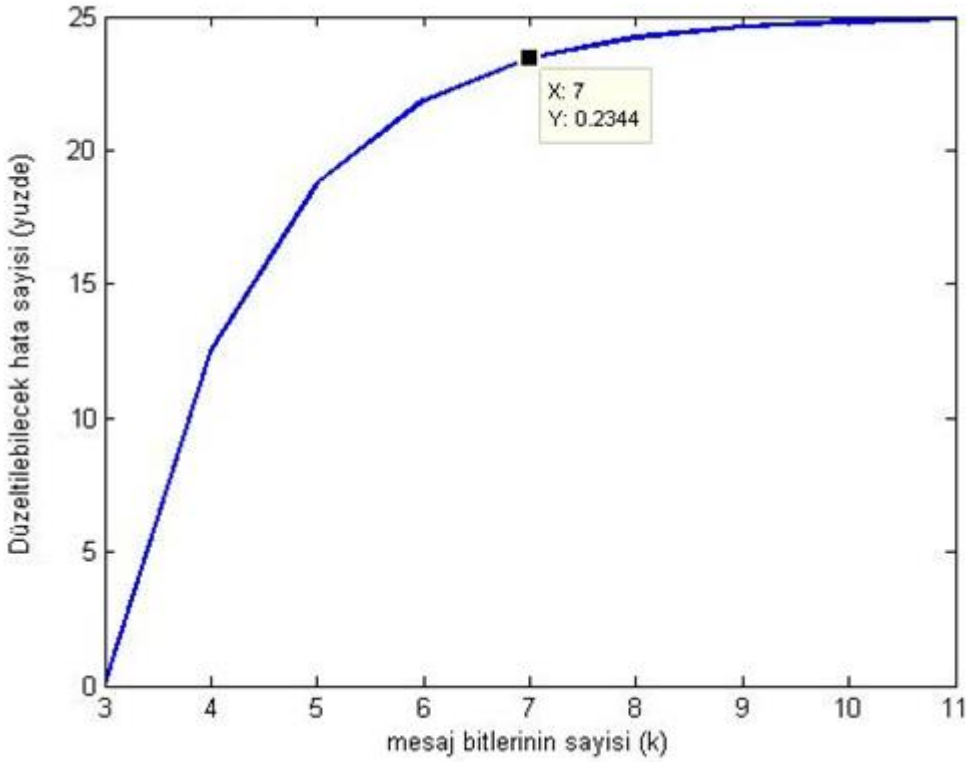


Şekil 3.3 : Rep işlemi.

Kodlama ve kod çözme sistemlerinin kolay tasarlanabilmesi bakımından birinci dereceden Reed Muller kodları tercih edilmiştir. Uygun Reed Muller kodunu seçebilmek için (n,k,d)=(2^m,m+1,2^{m-1}) kodunu seçebilmek için düzeltilebilecek hatalı bitlerin sayısı **t=** $\left\lfloor \frac{d-1}{2} \right\rfloor$ değerinin toplam kod uzunluğuna oranının ve 128 bitlik hatasız veri elde edebilmek için gereken PUF bitlerinin sayısı $n.\left\lceil \frac{s_k}{k} \right\rceil$ değerinin k ile değişimi incelenmiştir. İlgili grafikler Şekil 3.4 ve Şekil 3.5'te görülmektedir.



Şekil 3.4 : 128 bit mesaj için gereken PUF bitlerinin sayısı.



Şekil 3.5 : (2^{k-1},k,2^{k-2}) Reed Muller kodunun hata düzeltme kapasitesi.

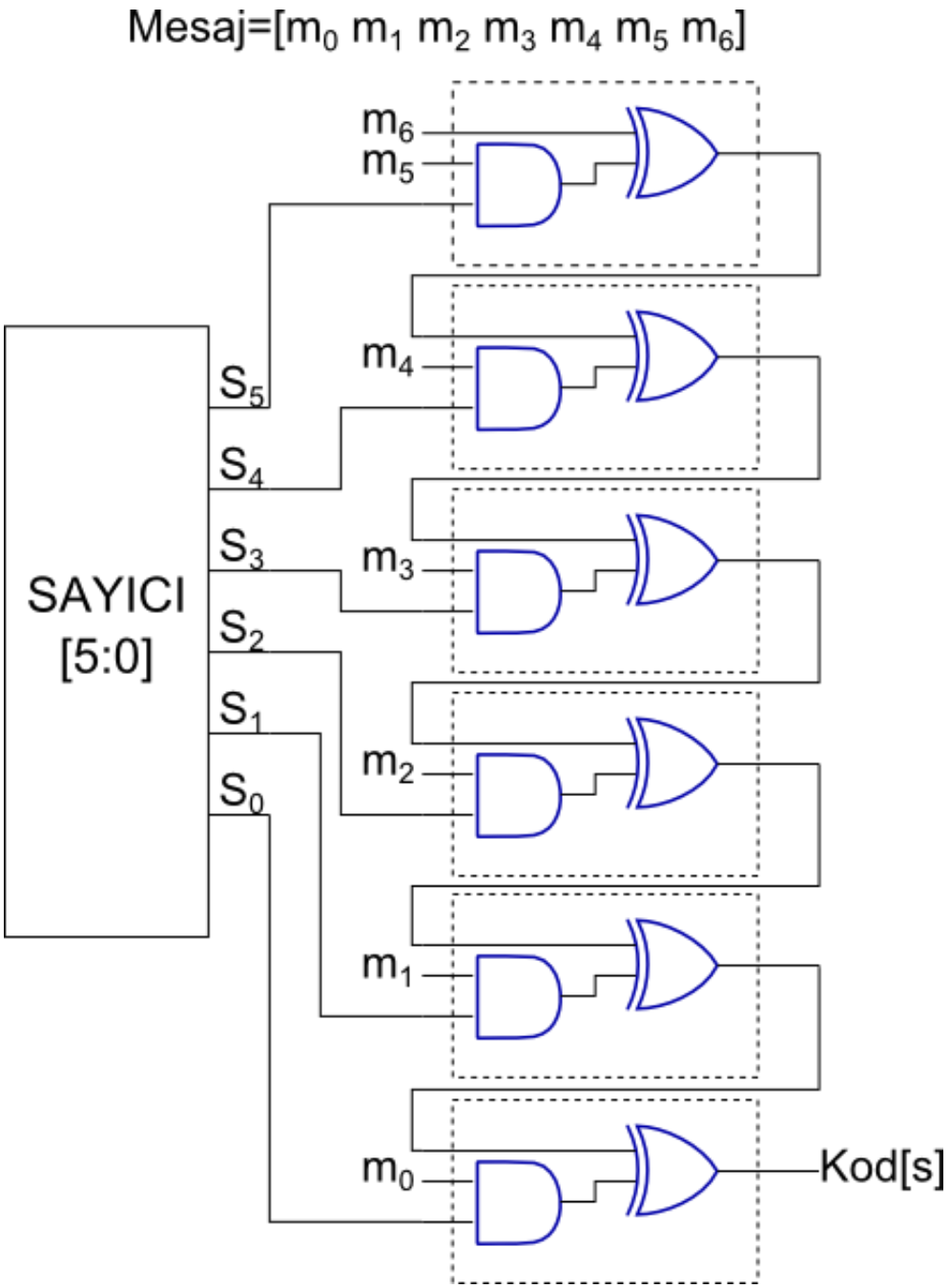
Buna göre düzeltilebilecek bit hata oranı %25' değerine yakınsamakta fakat k değeri arttıkça almamız gereken PUF bitlerinin sayısı üstel olarak artmaktadır. PUF bitlerinin sayısının artması alan olarak PUF tasarımının büyümesi anlamına gelmektedir. Bu analizden yola çıkarak k=7 değerine ait (64,7,32) Reed Muller kodunun seçilmesi uygun görülmüştür. Bu kod ile 15 hata düzeltilebilmekte bu da Gen işleminin her adımında alınan her 64 bitlik PUF verisi için %23.44 bit hata oranına karşılık gelmektedir. Buna göre 128 hatasız bit elde etmek için toplam 1216 adet PUF bitine ihtiyaç duyulmakta ve bu da Gen ile Rep işleminde 19 adım kullanılacağı anlamına gelmektedir.

3.4 Hata Düzeltici Sistemin Gerçeklenmesi

Hata düzeltici sistemi gerçeklemek üzere Reed-Muller kodlayıcı, Reed-Muller kod çözücü ve bu blokları kullanarak Gen ve Rep işlemlerini gerçekleştiren bloklar hem Matlab üzerinde yazılımsal olarak hem de Verilog donanım tanımlama dili ile donanım olarak gerçeklenmiştir. Daha önce bahsedilen halka osilatörlü PUF sistemi ile iki farklı FPGA üzerinden PUF verileri alınmıştı. Alınan PUF verilerindeki hataların düzeltilip düzeltilmediği Matlab üzerinde kontrol edilmiştir.

3.4.1 Reed-Muller kodlayıcı

Hata düzeltici sistemde kullanılmak üzere seçilen RM(1,6)=(64,7,32) koduna ait kodlayıcı sistem gerçeklenmiştir. Kodlayıcı mimarisi Şekil 3.6'da görülmektedir. Kodlama işlemi mesaj vektörü ile üretici matrisin modulo 2 olarak çarpımından ibarettir. İlk satır hariç tutulduğunda üretici matrisin sütunlarının sağdan sola doğru 0'dan 63'e sayma işlemi ile elde edilebileceği görülebilir. Sayıcının her konumu ilk satırdaki elemanlar hariç üretici matrisin sütunlarından birini oluşturur. İlk satırdaki bütün elemanların değeri 1 olduğundan ilk 6 meaj bitinin sayıcı değeri ile bit bazında çarpımının (AND) son mesaj biti ile toplamı (XOR) sayıcı değerinin gösterdiği kod bitinin değerini vermektedir.



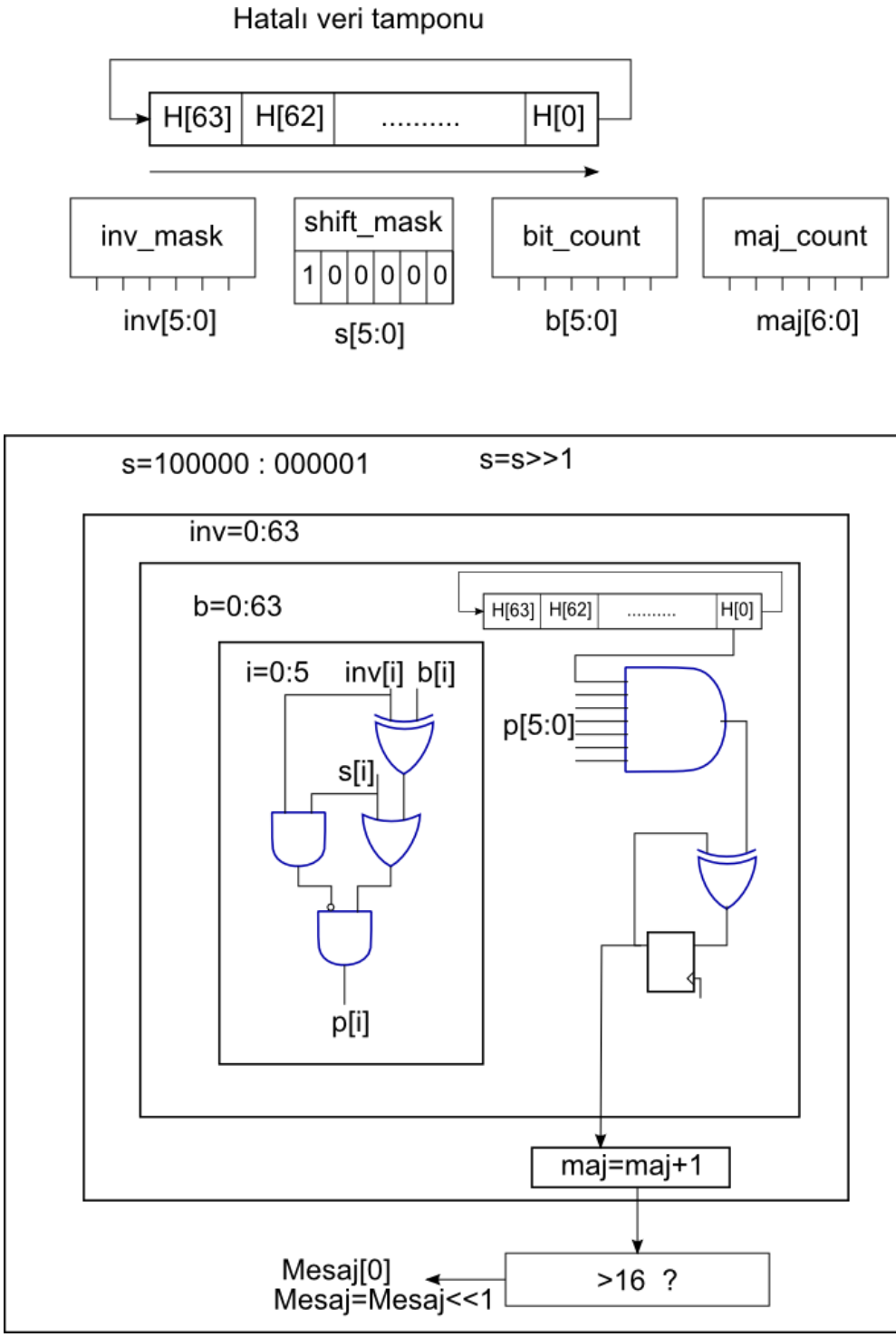
Şekil 3.6 : Reed Muller kodlayıcı.

Kodlayıcı FPGA üzerinde toplam 46 slice, 73 flip flop ve 24 LUT yer kaplamaktadır.

3.4.2 Reed-Muller kod çözücü tasarımı

RM(1,6)=(64,7,32) koduna ait kod çözücü gerçeklenmiştir. Üretici matrisin her bir satırı için önce karakteristik vektörler bulunur. Daha sonra karakteristik vektörler ile hata içeren kod vektörünün iç çarpımı hesaplanır. Çıkan iç çarpım sonuçlarının çoğunluk değeri üretici matrisin ilgili satırına karşılık gelen mesaj bitini vermektedir. Kod çözücü tasarımında karakteristik vektörlerin hesaplanması için evirme maskesi

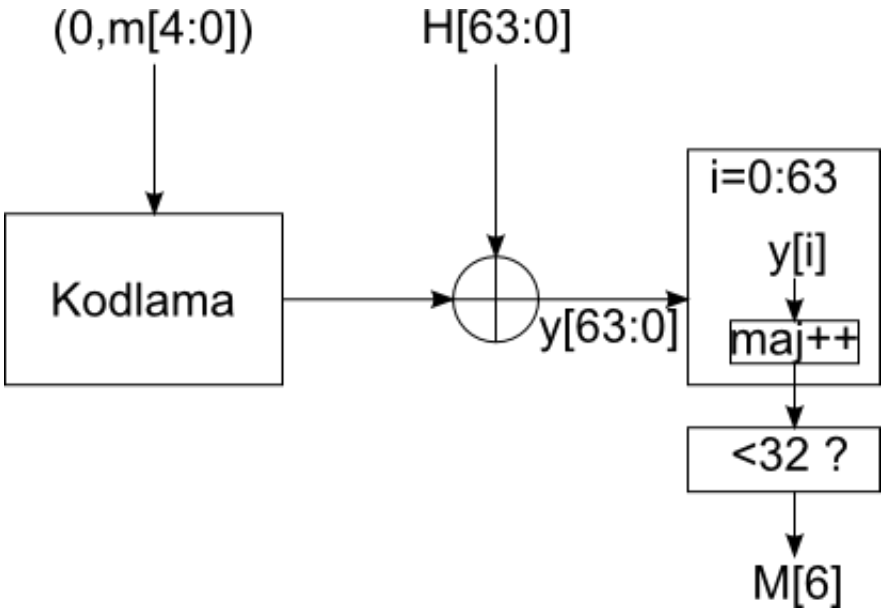
(inv_mask) sayıcısı ve kaydırma maskesi olarak kullanılan n'de bir kaydedici (shift mask one of n register); üretici matrisin sutunlarını göstermek üzere bit sayıcısı (bit_count) ve son olarak çoğunluk değerini hesaplamak üzere çoğunluk sayıcısı (maj_count) olmak üzere 4 farklı sayıcı kullanılmıştır. Kod çözücü sistemin bir bölümü Şekil 3.7'de gösterilmektedir.



Şekil 3.7 : Reed Muller kod çözücüsü.

Bu bölümde mesajın sadece ilk 6 biti belirlenmektedir. Üretici matrisin sütunları b[i] değeri ile belirlenmekte, shift_mask değeri ile hangi üretici matristeki 6 adet $\mathbf{x_i}$ satırından hangisi için karakteristik vektörlerin bulunacağı seçilmekte, inv[i] değeri ile de karakteristik vektör bulunurken birbiriyle çarpılan üretici matris satırlarından (**x**) hangisinin eviriğinin alınacağı belirlenmektedir. p[i] değeri karakteristik vektörü oluşturmak üzere kullanılan $\mathbf{x_i}$ vektörlerinin ilgili bitini göstermektedir. p[i] ile H[b] değerinin AND işleminden geçirilmesi karakteristik vektör ile hatalı verinin iç çarpım değerini hesaplamak üzere toplanması gereken ilgili biti göstermektedir. Bu bitler her b için XOR işlemi ile toplanarak iç çarpım değeri elde edilmektedir. İç çarpım değeri 0 ise maj_count bir arttırılmaktadır. Bütün inv değerleri yani bütün karakteristik vektörler için için iç çarpım değerleri elde edildikten sonra maj sayıcısı kontrol edilmektedir. Eğer maj sayıcısının değeri 16'dan büyükse iç çarpım değerlerinin çoğunluğu 0 olacağından ilgili mesaj bitinin gerçek değeri 0 olmaktadır. Tam tersinin gerçekleşmesi durumunda ilgili mesaj bitinin gerçek değeri 1 olmaktadır. Shift maskesi yardımıyla 7 adet mesaj bitinden üretici matrisin **x** satırlarıyla ilgili olan 6 tanesinin gerçek değeri belirlenir.

Son mesaj bitini belirlemek için öncelikle bu bit 0 olarak kabul edilir ve elde edilen 7 bitlik mesaj kodlanır. Elde edilen kod başlangıçtaki hatalı veri ile bit bazında XOR işleminden geçirilir. Elde edilen vektördeki bitlerin çoğunluk değeri son mesaj bitinin gerçek değerini vermektedir. Bu işlem Şekil 3.8'de gösterilmektedir.

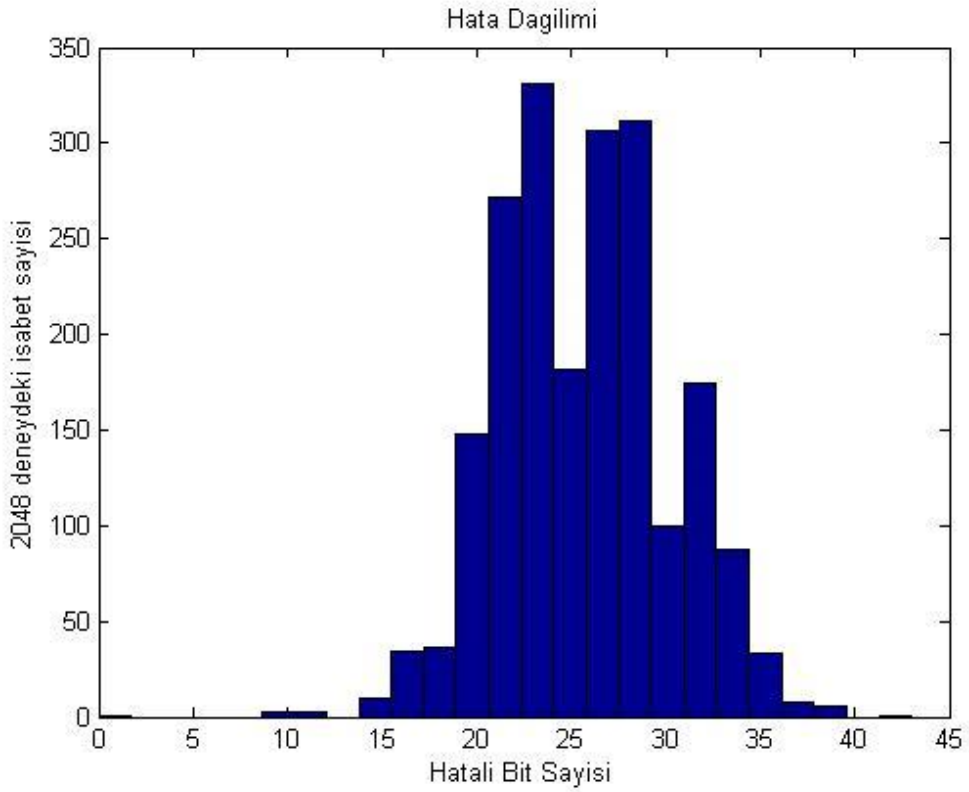


Şekil 3.8 : Son mesaj bitinin belirlenmesi.

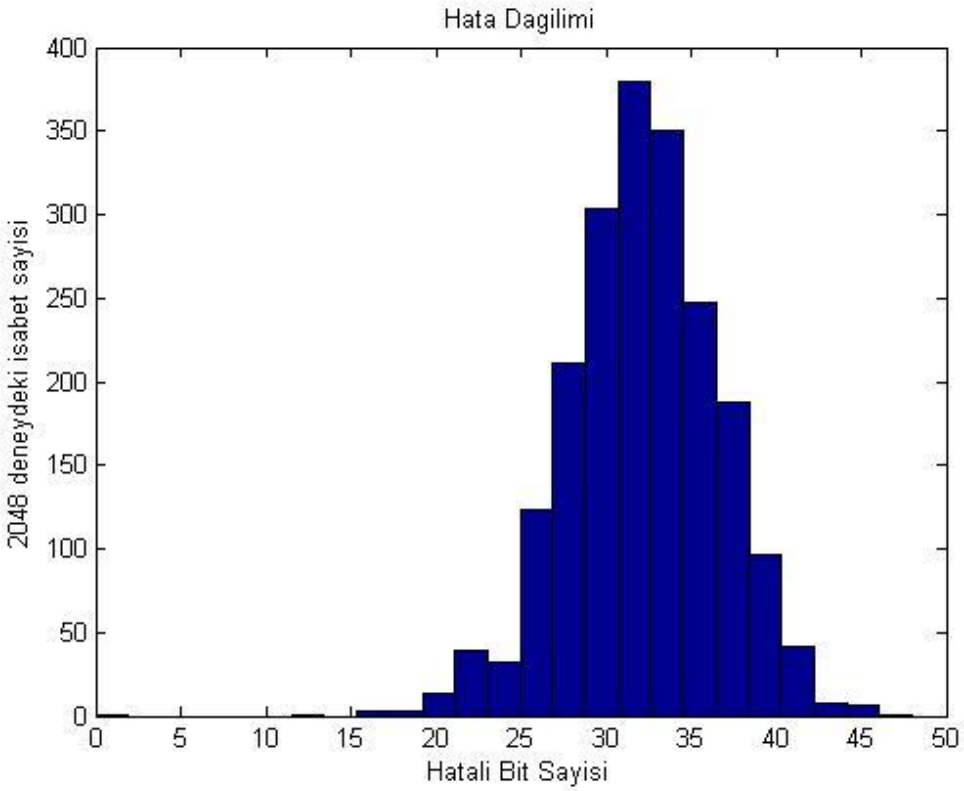
Kod çözücü FPGA üzerinde toplam 132 slice, 105 flip flop ve 239 LUT alan kaplamaktadır.

3.4.3 Test sonuçları

Hata düzeltme algoritması iki adet FPGA kartından alınan PUF verileri üzerinde Matlab kullanılarak denenmiştir. Toplam 2048 adet 1216 bitlik PUF verisi içerisinden bir tanesi referans seçilmiş, diğerleri üzerinde hata düzeltme algoritması koşturulmuştur. Örnek bir PUF referans değeri için diğer verilere ait hataların dağılımı 1. FPGA ve 2. FPGA için Şekil 3.9 ve Şekil 3.10'da görülmektedir.



Şekil 3.9 : Hata düzeltme öncesi 1. FPGA'daki PUF'a ait hata dağılımı.



Şekil 3.10 : Hata düzeltme öncesi 2. FPGA'daki PUF'a ait hata dağılımı.

İki farklı karttan alınan veriler üzerinde farklı farklı referans PUF değerleri seçilerek yapılan deneylerin tamamında tüm hataların başarıyla giderildiği görülmüştür.

4. KRİPTOGRAFİK ALT BLOKLARIN GERÇEKLENMESİ

Gerçeklenen PUF ve hata giderici sistem kullanılarak bir lisans doğrulama protokolü gerçeklenmiştir. Bu protokolde kullanılan kriptografik alt bloklar [9]'da bahsedilen LFSR tabanlı özet fonksiyonu ve RSA şifreleme ve imzalama bloklarıdır. RSA bloğunda kullanılan anahtarların bazı matematiksel koşulları sağlaması gerektiğinden PUF çıkışından elde edilen veri doğrudan anahtar olarak kullanılamamaktadır. Bu nedenle RSA bloğu içerisinde bir de açık ve gizli RSA anahtarı üretici blok tasarlanmıştır.

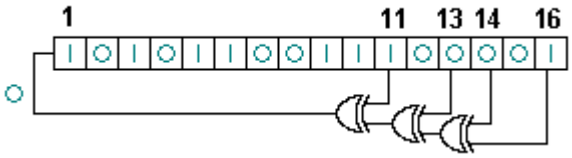
4.1 LFSR Tabanlı Özet Fonksiyonu

Hata kodlama sisteminin yapısından dolayı elde edilen 1216 hatasız PUF bitinde aslında 133 bitlik bir entropi olduğu gösterilmişti. Bu 1216 bitlik bit dizisindeki dağılımı uniform hale getirmek ve 1216 bitte bulunan entropiyi 128 bit içerisine toplamak amacıyla özet fonksiyonları kullanılmaktadır. Bu adım yardımcı bilgi hata düzeltme algoritmasında rastgelelik arttırma adımı olarak tanımlanmıştı. Ayrıca lisans doğrulama protokolü içerisinde doğrulama verisi olarak özet verileri kullanılmaktadır.

Özet fonksiyonu olarak kolay gerçeklenebilirlik bakımından [9]'da önerilen LFSR tabanlı hash fonksiyonunun kullanılması düşünülmüştür.

4.1.1 LFSR

LFSR lineer geribeslemeli ötelemeli kaydedici anlamına gelmektedir. Bu kaydedicinin giriş biti kaydedicinin önceki durumunun lineer bir fonksiyonudur. Lineer fonksiyon olarak XOR işlemi kullanılmaktadır. Kaydedicinin aldığı değerler rastgele görünümlüdür. Seçilen geribesleme fonksiyonuna göre kaydedicinin aldığı değerlerin tekrarlama periyodu çok uzun olabilmektedir. Bu nedenle sözde rastgele sayı üreteçlerinde kullanılırlar. Şekil 4.1'de örnek bir LFSR gösterilmektedir.



Şekil 4.1 : LFSR $x^{16}+x^{14}+x^{13}+x^{11}+1$.

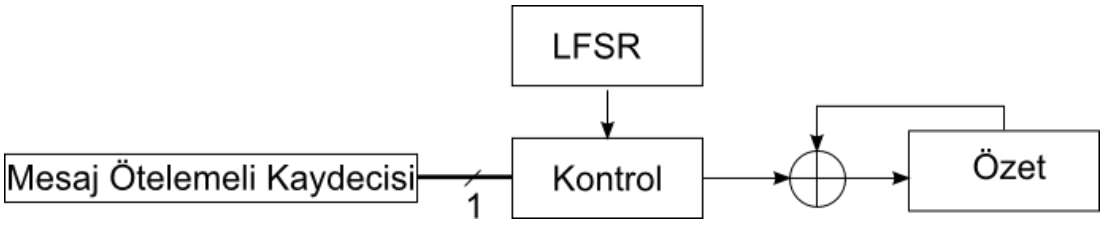
LFSR geribesleme fonksiyonları sonlu alan aritmetiği kullanılarak bir polinom olarak da gösterilir. Buna göre Şekil 4.1'deki LFSR $x^{16}+x^{14}+x^{13}+x^{11}+1$ polinomu ile ifade edilir. LFSR polinomunun primitif polinom seçilmesi halinde tekrarlama periyodu maksimum olur. Maksimum tekrarlama periyodu n bitlik bir kaydedicide 2^n-1 olur. Maksimum periyodu sağlayan polinomlara LFSR tablolarından bakılabilir [19].

Bu çalışmada LFSR kullanılan yerler ve bu LFSR bloklarına ait özellikler şunlardır:

- Hata düzeltme algoritmasında yardımcı veri üretimi sırasında rastgele kod üretmek üzere seçilen rastgele mesajın oluşturulması için 24 bitlik LFSR kullanılmıştır. Kullanılan primitif polinom $x^{24}+x^{23}+x^{21}+x^{20}+1$ polinomudur. Bu LFSR Verilog donanım tanımlama dili ile donanımsal olarak gerçeklenmiştir.
- Özet fonksiyonu içerisinde 128 bitlik bir LFSR kullanılmıştır. Bu LFSR bloğu da donanımsal olarak gerçeklenmiştir. Kullanılan primitif polinom $x^{128}+x^{127}+x^{126}+x^{121}+1$ polinomudur.
- RSA anahtarlarını üretirken gerekli iki adet büyük asal sayı üretilmesi gerekmektedir. 128 bitlik LFSR kullanılarak üretilen rastgele sayılara asallık testi yapılarak gerekli asal sayılar elde edilmektedir. Bu LFSR PUF verisini başlangıç değeri olarak almaktadır. RSA sistemleri Microblaze üzerinde gerçeklendiğinden bu LFSR bloğu yazılımsal olarak gerçeklenmiştir.

4.1.2 Özet fonksiyonu

LFSR tabanlı özet fonksiyonunun yapısı Şekil 4.2'de gösterilmiştir.

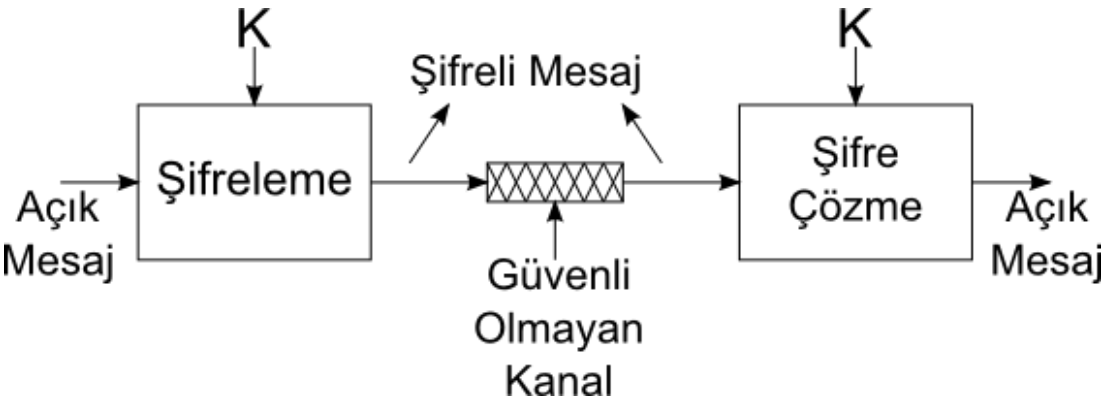


Şekil 4.2 : LFSR tabanlı özet fonksiyon.

Özeti alınacak veri bir ötelemeli kaydedicide bulunmaktadır. Her adımda verinin bir biti kontrol bloğu içerisine kaydırılmaktadır. Eğer gelen bit 1 ise "Özet" içeriği LFSR içeriği ile toplanmaktadır. Eğer gelen bit 0 ise "Özet" içeriği önceki değerini korumaktadır. Her adımda LFSR bir sonraki duruma geçmektedir. Bu yapıda LFSR değerinin her farklı başlangıç değeri için farklı bir özet fonksiyonu tanımlanabilir. Bu başlangıç değeri de PUF hata düzeltme algoritmasında bir yardımcı veri olarak kullanılabilir. Bahsedilen özet fonksiyonu PUF entropisini 128 bit içerisine toplamak ve lisans doğrulama protokolü içerisinde dongle ve ana sistem üzerinde doğrulama verisi oluşturmak için kullanılmıştır. PUF tarafında kullanılan özet fonksiyonu donanımsal olarak gerçeklenmiştir. Gerçeklenen özet fonksiyonu FPGA üzerinde 165 slice, 257 flip flop ve 186 LUT alan kaplamaktadır. Lisans doğrulama protokolü içerisinde ise bu özet fonksiyonu Microblaze işlemcisi üzerinde yazılımsal olarak gerçeklenmiştir.

4.2 RSA Algoritması

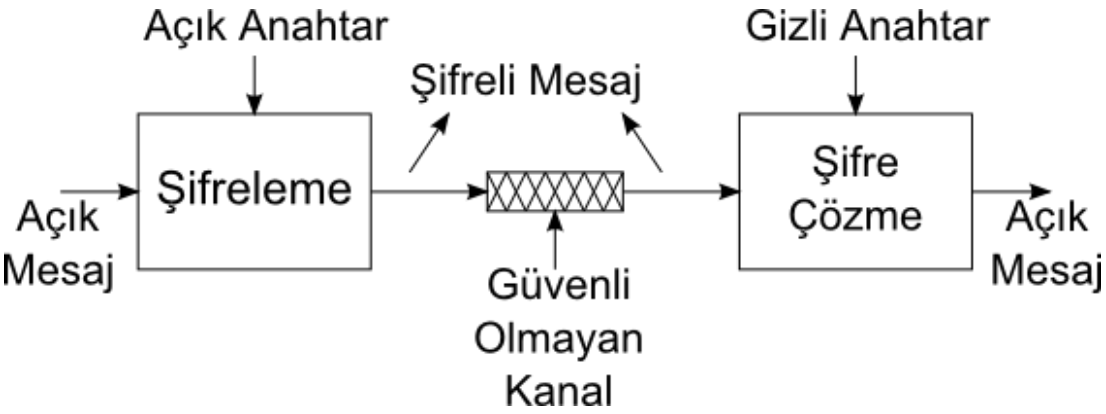
Önerilen lisans doğrulama sisteminde dongle ile ana sistem arasındaki haberleşmenin güvenli olabilmesi için bir kriptografik şifreleyici alt sisteme ihtiyaç duyulmaktadır. Kripto sistemleri simetrik anahtarlı ve açık anahtarlı sistemler olarak iki kategoriye ayrılabilir. Simetrik anahtarlı sistemlerde şifreleme ve şifre çözme işlemlerinde aynı anahtar kullanılır. AES algoritması simetrik anahtarlı şifrelemeye örnek olarak verilebilir.



Şekil 4.3 : Simetrik anahtarlı şifteleme.

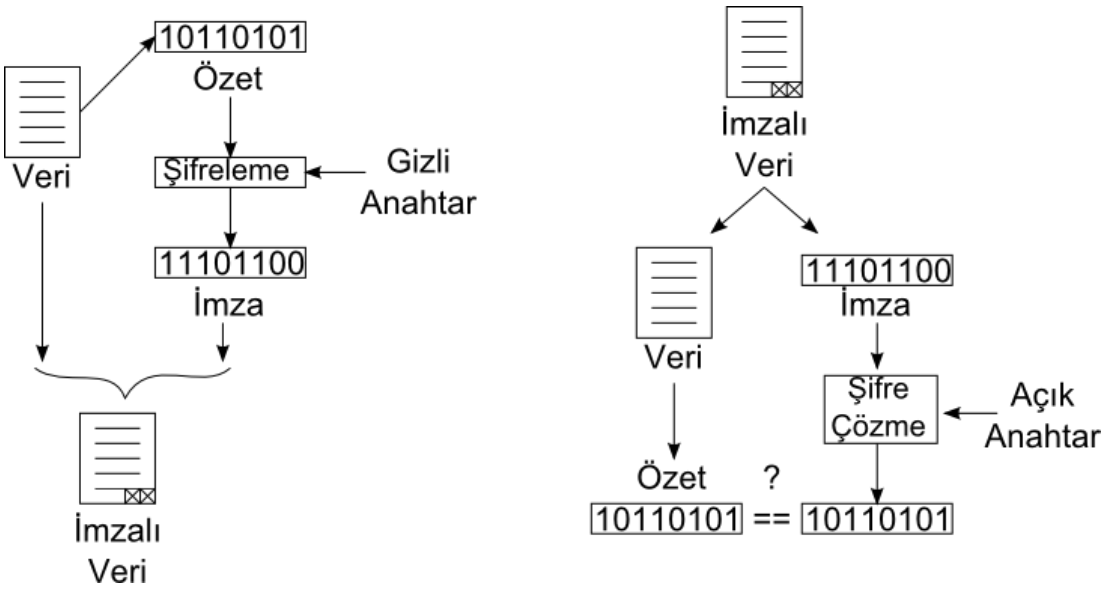
Açık anahtarlı sistemlerde ise bir açık bir de gizli anahtar bulunur. Şifreleme yapılmak istendiğinde alıcıya ait açık anahtar kullanılır. Alıcı kendi gizli anahtarıyla mesajı çözebilir. RSA algoritması açık anahtarlı bir şifreleme algoritmasıdır. Açık

anahtarlı sistemlerin güvenlik açısından avantajı şifreli mesajı açabilen anahtarın sadece alıcı sistemde bulunması ve sistem dışına çıkmamasıdır. Fakat gerçeklenmeleri simetrik anahtarlı sistemlere göre daha zordur. Açık anahtarlı sistemlerde özel koşulları sağlayan anahtar çiftlerinin üretilmesi bir problem olup, simetrik anahtarlı sistemlerde ise anahtarın alıcı ve gönderici arasında güvenli olarak paylaşılması bir problemdir.



Şekil 4.4 : Açık anahtarlı şifreleme.

Açık anahtar algoritmaları ile sayısal imzalama ve sayısal imza doğrulama sistemleri de oluşturulabilir. İmza oluşturmak için gönderilmek istenen verinin özeti (*hash*) çıkarılarak bu veri göndericiye ait gizli anahtarla şifrelenir. Şifreli veri özeti çıkarılmış veriye ait imza değeridir. İmzayı doğrulamak isteyen alıcı da verinin özetini hesaplar ve şifreli veriyi göndericiye ait açık anahtarla açar. Alıcı açılan veriyi kendi hesapladığı özet ile karşılaştırarak imzanın doğruluğunu kontrol eder.



Şekil 4.5 : İmzalama ve imza doğrulama.

Bu çalışmada, önerilen lisans doğrulama sistemi için açık anahtarlı RSA algoritması kullanılarak şifreleme, şifre çözme, imzalama ve imza doğrulama işlemleri gerçekleştirilmektedir. RSA algoritması Rivest, Shamir ve Adleman tarafından geliştirilmiştir [20]. Başta bankacılık olmak üzere birçok sektörde şifreleme ve sayısal imza sistemleri için kullanılmaktadır. Kullanılan anahtarların asimetrik olması ve sistemin güvenliği iki çok büyük asal sayının çarpımının çarpanlarına ayrılmasının zorluğundan ileri gelmektedir.

4.2.1 RSA anahtarları

RSA sistemi içerisinde bir gizli bir de açık anahtar bulunur. Açık anahtar herkes tarafından bilinebilir ve sadece şifreleme ile imza doğrulamada kullanılır. Gizli anahtar ise sadece ilgili taraf tarafından bilinebilir ve sadece şifre çözme ile imzalama işlemlerinde kullanılır. Gizli ve açık RSA anahtarları şu adımlar ile üretilir:

- İki adet aynı bit sayısına sahip asal sayı (p ve q) seçilir. Bu sayılar sistem güvenliği gereği rastgele seçilmelidir. Miller-Rabin testi gibi çeşitli asallık testleriyle bu sayılar kolaylıkla bulunabilir.
- p ve q sayılarının çarpımı (n=p.q) hesaplanır. n sayısı açık ve gizli anahtarlar için mod değeri olarak kullanılır. n sayısının bit uzunluğu RSA sisteminin uzunluğudur. Günümüzde en az 2048 bit uzunluğunun kullanılması önerilmektedir. Bu uzunluk arttıkça n sayısının çarpanlarına ayrılması güçleşmekte ve sistem güvenliği artmaktadır.
- $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1) = n - (p + q - 1)$ değeri hesaplanır.
- e ve $\varphi(n)$ aralarında asal (gcd(e, $\varphi(n)$)=1) ve $1<e<\varphi(n)$ olacak şekilde bir e sayısı seçilir. e ve n sayıları açık anahtarı oluşturur. e sayının bit sayısının az olması şifreleme işlemini kolaylaştırır. Bu sayı genelde 65537 olarak seçilir. 3 gibi daha küçük değerlerin seçilmesinin bazı durumlarda daha az güvenli olduğu gösterilmiştir [21].
- $d \equiv e^{-1} \pmod{\varphi(n)}$; hesaplanır. d sayısı e sayısının $\varphi(n)$ modunda çarpmaya göre tersidir. Bu değer genişletilmiş Öklid algoritması ile hesaplanabilir. d ve n sayıları gizli anahtarı oluşturur.
- Sistem güvenliği için d, p, q ve $\varphi(n)$ kesinlikle gizli tutulmalıdır.

4.2.2 RSA işlemleri

RSA algoritmasında şifreleme ve şifre çözme, imzalama ve imza doğrulama işlemleri modüler üs alma işlemleri ile gerçekleştirilir. Buna göre

- $0 \leq m < n$ olan bir m değerinden şifrelenmiş c verisini elde etmek için $m^e (mod\ n)$ işlemi gerçekleştirilir.
- c şifreli verisinden m verisini elde etmek için $c^d (mod\ n)$ işlemi gerçekleştirilir.
- s imzasını doğrulamak üzere h özetini elde etmek için $s^e (mod\ n)$ işlemi gerçekleştirilir.
- h özet verisinden s imzasını elde için $h^d (mod\ n)$ işlemi gerçekleştirilir.

4.2.3 RSA sisteminin gerçeklenmesi

Tüm RSA işlemleri Microblaze işlemcisi üzerinde yazılımsal olarak gerçeklenmiştir. Büyük sayılarla yapılan çarpma işlemi, Miller-Rabin asallık testi, modüler üs alma gibi işlemler için BigDigits [10] kütüphanesi kullanılmıştır. Rastgele sayı üretimi için ise 128 bitlik yazılımsal LFSR gerçeklemesi yapılmıştır. Kullanılan BigDigits fonksiyonları şunlardır:

- mpRabinMiller: Olasılıksal asallık testi yapan Miller Rabin algoritmasını gerçekleyen fonksiyondur. Bu fonksiyon ile PUF'tan veriyi başlangıç değeri olarak LFSR bloğundan alınan 128 bitlik rastgele sayıların asallık kontrolü yapılır. Bulunan ilk iki asal sayı p ve q olarak seçilir.
- mpMultiply: Büyük sayılarda çarpma fonksiyonudur. Anahtar üretiminde p ve q sayılarının çarpımında kullanılır.
- mpModInv: Modüler olarak çarpmaya göre tersi hesaplar. Anahtar üretiminde e sayısının mod n'deki tersi d sayısının bulunmasında kullanılır.
- mpModExp: Büyük sayılarda modüler üs alma işlemini gerçekleştirir. Bu fonksiyon şifreleme, şifre çözme, imzalama ve imza doğrulama işlemlerinde kullanlır.

RSA alt sistemi bahsedilen BigDigits fonksiyonlarını kullanarak şu adımlar ile gerçeklenmiştir.

- 128 bitlik yazılımsal LFSR bloğu başlangıç değeri PUF değeri olarak seçilerek p ve q asal sayılarını üretmek üzere kullanılmıştır.
- Her bir LFSR değeri için mpRabinMiller fonksiyonu ile asallık testi yapılmıştır. LFSR sürekli çalıştırılmış ve karşılaşılan ilk iki asal sayı p ve q değeri olarak seçilmiştir.
- mpMultiply fonksiyonu kullanılarak p.q çarpımı hesaplanmış ve n değeri belirlenmiştir. Ayrıca yine aynı fonksiyon ile $\varphi(n)=(p-1)(q-1)$ çarpımı hesaplanmıştır.
- e anahtarı 65537 olarak seçilmiştir.
- d anahtarı mpModInv fonksiyonuyla e'nin $\varphi(n)$ modunda çarpmaya göre tersi hesaplanarak belirlenmiştir.
- Son olarak yapılması gereken imzalama, şifreleme gibi işlemleri gerçeklemek için RSA işlemleri bölümünde belirtilen modüler üstel işlem gerçeklenmiştir. Bu işlem için mpModExp fonksiyonu kullanılmıştır.

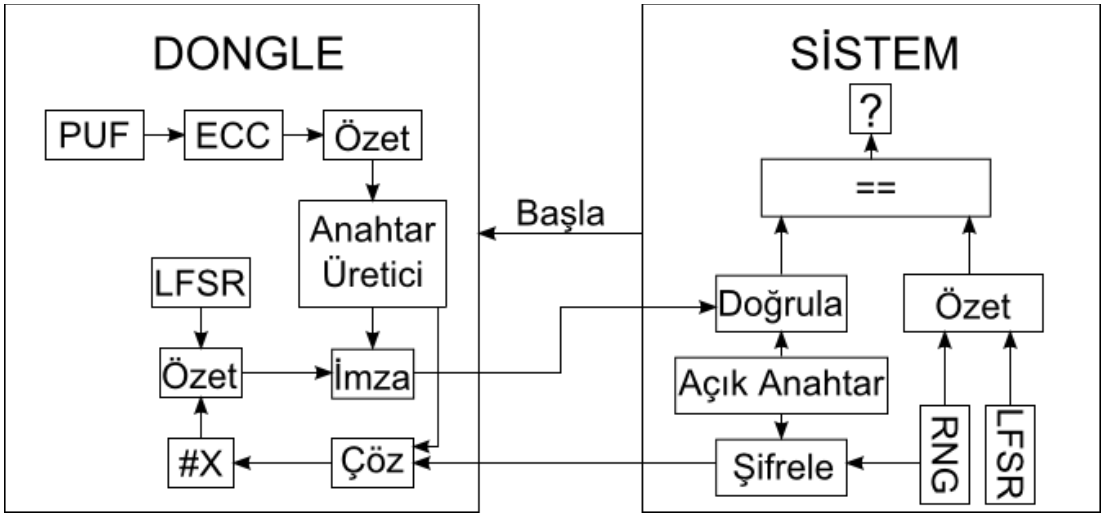
Lisans doğrulama protokolüne geçmeden önce RSA kriptografik alt sistemi ayrıca test edilmiştir. Microblaze işlemcisi üzerinde gerçeklenen test sistemi ile tanımlanan LFSR başlangıç değeri için RSA anahtarları üretilmiştir. Ayrıca tanımlanan bir mesaj şifrelenmiş ve şifresi çözülerek aynı mesaj başarıyla tekrar elde edilmiştir. Sonuçlar bilgisayar üzerinden UART yolu ile izlenmiştir.

5. LİSANS DOĞRULAMA PROTOKOLÜNÜN GERÇEKLENMESİ

Tasarlanan anahtar üretici blok, RSA bloğu ve diğer kriptografik alt bloklar yardımıyla bir lisans doğrulama protokolünün gerçeklenmesi amaçlanmıştır. Bu amaçla öncelikle bir lisans doğrulama protokolü önerilmiştir. Bu protokol içerisindeki tüm alt bloklar gerçeklenerek test edildikten sonra önerilen lisans doğrulama protokolü FPGA üzerinde doğrulanmıştır.

5.1 Protokol Mimarisi

Lisans doğrulama sistemi bir dongle bir de ana sistemden oluşmaktadır. Ana sistem kendisinin lisanslı olup olmadığını kontrol etmek için doğru dongle donanımı ile iletişime geçmelidir. Önerilen lisans doğrulama sisteminin yapısı Şekil 5.1'de görülmektedir.



Şekil 5.1: Lisans doğrulama sisteminin yapısı.

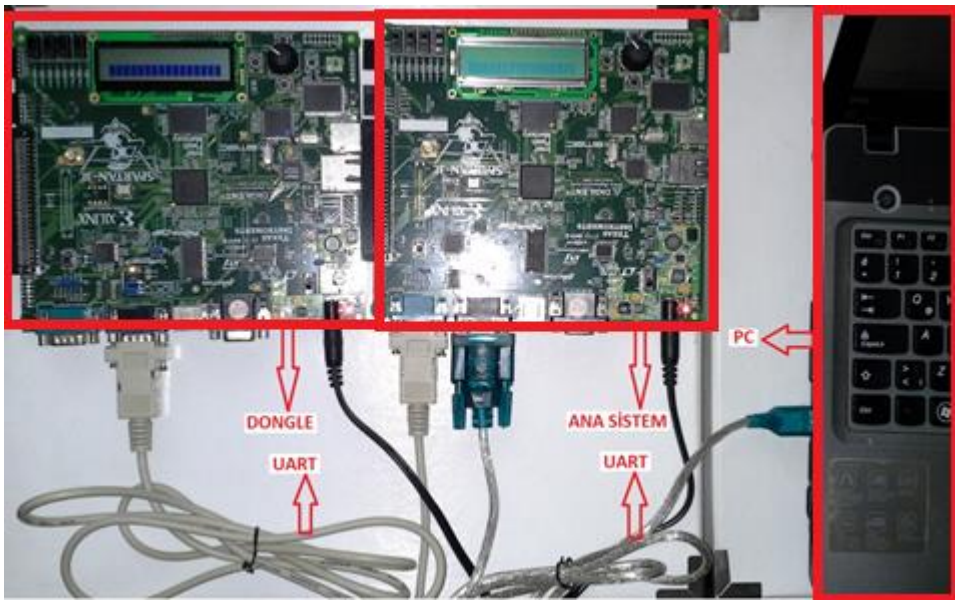
Önerilen protokole göre ana sistem gerçek fonksiyonunu yerine getirmeye başlamadan önce doğru dongle donanımının takılı olup olmadığını şu adımları izleyerek kontrol eder:

1) Sistem rastgele bir sayı üretir.

2) Sistem rastgele sayıyı açık anahtar ile şifreler ve bunu "BAŞLA" işareti ile birlikte dongle donanımına gönderir.
3) Dongle PUF verisini okur ve hata düzeltme işlemleri yapar.
4) Dongle hatasız PUF verisinin özetini hesaplar.
5) Dongle PUF özetini kullanarak gizli anahtarı üretir.
6) Dongle şifrelenmiş rastgele sayıyı gizli anahtar ile çözer.
7) Dongle ana sistem ile her doğrulama yapıldığında senkron olarak ilerleyen LFSR verisi ve rastgele sayının özetini hesaplar.
8) Dongle hesaplanan özeti gizli anahtar ile imzalar ve bunu ana sisteme gönderir.
9) Ana sistem dongle ile senkron olan LFSR verisi ve daha önce ürettiği rastgele sayının özetini hesaplar.
10) Ana sistem dongle donanımından gelen imzalı veriyi açık anahtar ile çözer.
11) Ana sistem çözülen imza verisi ile kendi hesapladığı özet verisinin birbirine eşit olup olmadığını kontrol eder. Eşit ise çalışma izni verilir.

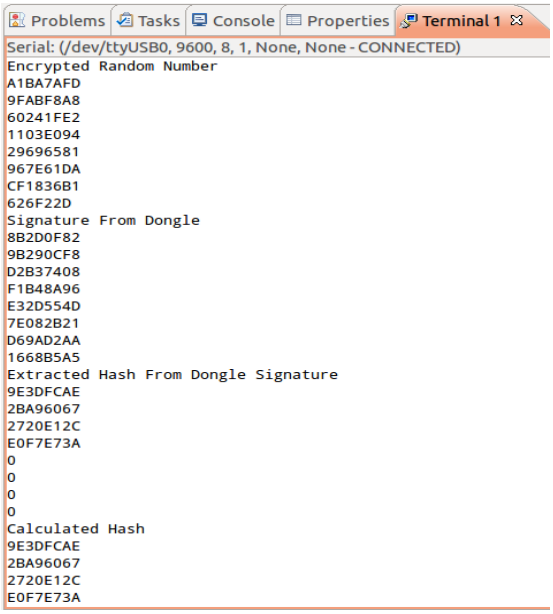
5.2 Protokolün Gerçeklenmesi

İki farklı FPGA kullanılarak bu protokolün fonksiyonelliği gösterilmiştir. Kullanılan FPGA'lardan birinde dongle donanımı gerçeklenmiş, diğerinde ise dongle donanımı ile iletişim kurarak lisansı doğrulayan bir ana sistem oluşturulmuştur. Ana sistemin bulunduğu FPGA'daki lisans doğrulama adımları ayrıca bilgisayar yardımıyla izlenmiştir. Sistem bileşenleri Şekil 5.2'de gösterilmektedir. Ana sistem tamamıyla Microblaze işlemcisi kullanılarak yazılımsal olarak gerçeklenmiştir. Ana sistem bilgisayardan yardımcı veriyi güncelleme, açık anahtarı alma ve doğrulamayı başlatma olmak üzere 3 farklı komut almaktadır. Yardımcı veri komutu geldiğinde ana sistem dongle donanımına bu komutu aynen iletir ve dongle PUF verisini düzeltmek için kullandığı yardımcı veriyi günceller. Açık anahtar komutu gönderildiğinde ise ana sistem dongle donanımından açık anahtarı ister. Dongle önce PUF verisini okur ve hata düzeltme işlemi yapar. Daha sonra hatasız PUF verisinin özetini hesaplar ve bunu RSA anahtarı üretmek amacıyla bir başlangıç değeri (*seed*) olarak kullanır.



Şekil 5.2: Lisans doğrulama sisteminin bileşenleri.

RSA anahtarı üretiminde kullanılan asal sayılar belirlenir ve gerekli hesaplamalar yapılarak açık anahtar üretilir. Son olarak bu değer ana sisteme gönderilir. Açık anahtarın güvenli bir ortamda bir defa okunarak ana sistemde saklanması da mümkündür. Burada fonksiyonellik gösterilmeye çalışıldığından açık anahtar UART üzerinden istenildiği zaman okunabilmektedir. Açık anahtarı alan ana sistem bu değeri saklar ve bir kopyasını bilgisayara gönderir. Bilgisayardan ana sisteme doğrulamayı başlat komutu gönderildiğinde ise ana sistem dongle donanımına doğrulama yapmak istediğini bildiren komutu gönderir. Ardından rastgele bir sayıyı daha önce elde ettiği açık anahtarı kullanarak şifreler ve bunu da dongle donanımına gönderir. Dongle PUF verisini okuyarak açık ve gizli anahtar çiftini tekrar oluşturur. Gizli anahtar ile şifre çözme işlemi yaparak rastgele sayıyı elde eder ve iki sistemde ortak ilerleyen LFSR değeri ve rastgele sayının özetini hesaplar. Elde ettiği özeti gizli anahtar ile imzaladıktan sonra bunu ana sisteme geri gönderir. Ana sistem imzayı doğrulayarak dongle donanımından gelen özeti elde eder ve kendi içerisinde bulunan rastgele sayı ve LFSR değerlerinden kendi özetini hesaplar. Kendi hesapladığı özeti ve kendisine gelen özeti karşılaştırarak doğrulamayı gerçekleştirir. Bütün bu veriler ana sistemden bilgisayara da gönderilir ve bilgisayar üzerinden sistemin fonksiyonelliği test edilmiş olur. Yapılan doğrulama işlemi sonucu bilgisayardan okunan örnek bir veri Şekil 5.3'te gösterilmiştir.



Şekil 5.3: Elde edilen doğrulama verileri.

İmzadan elde edilen özet ile hesaplanan özet uyumlu olduğundan doğrulama başarılıdır.

6. SONUÇ VE ÖNERİLER

Çalışmada bir PUF tabanlı bir anahtar üretici ve bu anahtar kullanılarak bir lisans doğrulama protokolü gerçeklenmiştir. Bu sistemde PUF, yardımcı verilerle hata düzeltici, RSA anahtarı üretici, RSA şifreleme ve RSA imza blokları bulunmaktadır. PUF bloğu halka osilatörü, Lehmer ve Gray kodlama bloklarından; hata düzeltme bloğu Reed Muller kodlayıcı ve kod çözücü bloklarından ve bir LFSR tabanlı özet fonksiyonundan oluşmaktadır. Bu bloklar FPGA üzerinde gerçeklenmiş ve PUF verileri incelenerek hata düzeltme sisteminin PUF verilerindeki hataları düzettiği gösterilmiştir. RSA şifreleme ve imza algoritmaları da FPGA içerisinde bulunan Microblaze işlemcisi üzerinde yazılımsal olarak gerçeklenmiştir. Tüm alt blokların kullanılmasıyla bahsedilen lisans doğrulama protokolünü uygulayan bir dongle gerçeklenmiştir. Başka bir FPGA üzerinde ise dongle ile iletişime geçen bir lisans doğrulayıcı ana sistem gerçeklenmiştir. Bahsedilen tüm bloklar ve FPGA üzerinde kaplanılan alan Çizelge 6.1'de gösterilmektedir.

Çizelge 6.1 : Sistem bileşenlerinin Spartan 3E FPGA üzerinde kapladığı alan.

Bileşen	Slice	Flip Flop	Lut	Bellek [kbit]
PUF	201	384	384	
Lehmer & Gray	285	95	544	
LFSR Özet	165	257	186	
Reed Muller Kodlayıcı	46	73	24	
Reed Muller Kod Çözücü	132	105	239	
Anahtar Üretici (Toplam)	978	1072	1893	1.2
RSA+UART+Microblaze	1459	1031	2165	14
Dongle	2944	2766	4258	18
Ana Sistem	1617	1823	2308	17

Özellikle Microblaze işlemcisinin RSA gerçeklemesinde kullanılmasının tasarım alanını arttırdığı söylenebilir. Kriptografik işlemleri yapmaya elverişli çok az alan kaplayan işlemci blokları ile büyük sayılarda işlem yapan yazılım kütüphanesi oluşturularak RSA sistemlerinin gerçeklenmesi üzerine çalışılabilir. Daha sonraki çalışmalarda yardımcı verilerle hata düzeltme algoritmalarının ve önerilen lisans doğrulama protokolünün güvenlik açıklarının incelenmesi önerilmektedir. Özellikle yardımcı verilerin açığa çıkardığı bilgi üzerinden yapılabilecek ataklar konusunda

çalışılabilir. Ayrıca PUF'un farklı koşullarda ve daha fazla sayıda çip üzerinde karakterize edilmesi, bahsedilen sistemin alan olarak iyileştirilmesi ve tasarımcılar ile müşteriler için olası kullanım senaryolarının belirlenmesi üzerinde çalışılabilir.

KAYNAKLAR

[1] **Govem, B.** (2013). Klonlanamaz Fonksiyonlar ve Yardımcı Bilgiler Kullanılarak Patent Hakları Korunması, Özgün Algoritma ve Donanımın Güvenliğinin Sağlanması, *İstanbul Teknik Üniversitesi (İTÜ).*

[2] **Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.** (2007). Physical Unclonable Functions and Public Key Crypto for FPGA IP Protection. *In: International Conference on Field Programmable Logic and Applications - FPL, IEEE 189-195*

[3] **Handschuh, H., Schrijen, G. J., Tuyls, P.** (2010). Hardware Intrinsic Security from Physically Unclonable Functions, *Towards Hardware-Intrinsic Security, Information Security and Cryptography, pages 39-53. Springer Berlin Heidelberg*

[4] **Bösch, C., Guajardo, J., Sadeghi, A.R., Shokrollahi, J. ve Tuyls, P.** (2008). Efficient Helper Data Key Extractor on FPGAs. *In: Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems (CHES). Lecture Notes in Computer Science, vol. 5154, pp. 181–197. Springer-Verlag*

[5] **Maes, R., Herrewege, A.V., Verbauwhede, I.** (2012). PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator. In: Workshop on Cryptographic Hardware and Embedded Systems (CHES). (2012) 302–319

[6] **Maes, R., Tuyls, P., Verbauwhede, I.** (2009). Soft Decision Helper Data Algorithm for SRAM PUFs. *In: IEEE Symposium on Information Theory (ISIT). 2101–2105*

[7] **Yu, M.D.M., M'Raihi, D., Sowell, R., Devadas, S.** (2011). Lightweight and Secure PUF Key Storage Using Limits of Machine Learning. *In: Proceedings of the 13th International Workshop on Cryptographic Hardware and Embedded Systems (CHES). Lecture Notes in Computer Science, vol. 6917, pp. 358–373*

[8] **Xin, X., Kaps, J., Gaj, K.** (2011). A Configurable Ring-Oscillator Based PUF for Xilinx FPGAs. *Proc. 14th EUROMICRO Conference on Digital System Design DSD'11, pp 651–657, IEEE*

[9] **Krawczyk, H.** (1994). LFSR-based Hashing and Authentication. *In Desmedt, Y., ed.: Advances in Cryptology - CRYPTO '94. Volume 839 of LNCS., Springer 129-139*

[10] Big Digits Library, http://www.di-mgt.com.au/bigdigits.html, alındığı tarih: 13.09.2014

[11] **Simpson, E., Schaumont, P.** (2006). Offline Hardware/Software Authentication for Reconfigurable Platforms. *Cryptographic Hardware and Embedded Systems - CHES. Volume 4249 of LNCS., Springer 311-323*

[12] **Gora, M. A., Maiti, A., Schaumont P.** (2010). A flexible design flow for software IP binding in FPGA, *IEEE Trans. Ind. Inf., vol. 6, no. 4, pp.719–728*

[13] **Maes, R., Verbauwhede, I.** (2010). Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. *Towards Hardware-Intrinsic Security. Information Security and Cryptography. Springer 3–37*

[14] **Kaya, G.** (2012). Reliability Enhancement of Ring Oscillator Based Physically Unclonable Functions, *İstanbul Teknik Üniversitesi (İTÜ)*

[15] **Kumar, S.S., Guajardo, J., Maes, R., Schrijen, G.J., Tuyls, P.** (2008). Extended abstract: The butterfly PUF protecting IP on every FPGA. *In IEEE International Workshop on Hardware-Oriented Security and Trust, 2008. HOST 2008, pages 67-70*

[16] **Anderson, J.** (2010). A PUF design for secure FPGA-based embedded systems. In Proceedings of the 2010 Asia and South Pacific Design Automation Conference (pp. 1-6). IEEE Press.

[17] **Maiti, A., Schaumont, P.** (2011). Improved Ring Oscillator PUF: An FPGA-friendly Secure Primitive, *IACR Journal of Cryptology 24 375–397*

[18] **Cooke B**. (1999). Reed Muller Error Correcting Codes, *MIT Undergraduate Journal of Mathematics*

[19] **Ward, R., Molteno, T.** (2007). Table of Linear Feedback Shift Registers, *Department of Physics, University of Otago*

[20] **Rivest, R.L., Shamir, A. and Adleman, L.**, (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, 21(2), 120–126.

[21] **Boneh, D.** (1999). Twenty Years of Attacks on the RSA Cryptosystem. *Notices of the American Mathematical Society, 46(2):203–213.*

ÖZGEÇMİŞ

Ad-Soyad **:** Şahin BAŞ

Doğum Tarihi ve Yeri: 1989, İstanbul

E-Posta **:** bassa@itu.edu.tr

Lisans **:** 2011, İstanbul Teknik Üniversitesi, Elektronik Mühendisliği